A Crowd-Sensing Framework for Allocation of Time-Constrained and Location-Based Tasks

Rebeca Estrada, Rabeb Mizouni, Hadi Otrok, *Senior Member, IEEE*, Anis Ouali, and Jamal Bentahar, *Member, IEEE*

Abstract—Thanks to the capabilities of the built-in sensors of smart devices, mobile crowd-sensing (MCS) has become a promising technique for massive data collection. In this paradigm, the service provider recruits workers (i.e., common people with smart devices) to perform sensing tasks requested by the consumers. To efficiently handle workers' recruitment and task allocation, several factors have to be considered such as the quality of the sensed data that the workers can deliver and the different tasks locations. This allocation becomes even more challenging when the MCS tries to efficiently allocate multiple tasks under limited budget, time constraints, and the uncertainty that selected workers will not be able to perform the tasks. In this paper, we propose a service computing framework for time constrained-task allocation in location based crowd-sensing systems. This framework relies on (1) a *recruitment algorithm* that implements a multi-objective task allocation algorithm based on Particle Swarm Optimization, (2) *queuing schemes* to handle efficiently the incoming sensing tasks in the server side and at the end-user side, (3) a *task delegation mechanism* to avoid delaying or declining the sensing requests due to unforeseen user context, and (4) a *reputation management component* to manage the reputation of users based on their sensing activities and task delegation. The platform goal is to efficiently determine the most appropriate set of workers to assign to each incoming task so that high quality results are returned within the requested response time. Simulations are conducted using real datasets from Foursquare[1] and Enron email social network.[2] Simulation results show that the proposed framework maximizes the aggregated quality of information, reduces the budget and response time to perform a task and increases the average recommenders' reputation and their payment.

Index Terms—Mobile crowd sensing, worker selection, particle swarm optimization (PSO)

1 Introduction

Mobile crowd-sensing (MCS) is a new paradigm in which a crowd of ordinary citizens utilize their mobile phone or smart devices to conduct complex and large-scale sensing tasks [1]. The user mobility makes MCS a versatile platform that can replace or complement current static sensing infrastructures. MCS systems benefit several applications in various areas such as community dynamics monitoring (i.e., traffic planning [2], environment monitoring [3], or public safety [4]).

The main components of an MCS system are: task manager, customers, and workers. The "workers" are enlisted to perform tasks in return for some compensation or incentive (e.g., entertainment, service, and money) [5]. The customers are the sensing task initiators. Each sensing task has its own requirements (e.g., deadline and budget), and is published on the platform to recruit mobile users to perform it. The task manager is the MCS platform that usually allocates the sensing tasks to appropriate workers.

MCS systems rely on user-contributed or crowd-source information. In other words, a task may be answered by one or multiple workers, depending on the application domain and the task requirements. Some platforms require a single user to perform a task while in others, such as Gigwalk,[3] many users are required to answer the task request to ensure the reliability of the collected information. In the particular case of location-based and time-sensitive sensing tasks, such as checking the on-shelf availability of a product in a convenience store, the users can collect the data at the precise time and location [6]. With this information, any company can reduce the cost of taking inventories, while maintaining the proper stock levels at different stores. Currently, several well-known brands and retailers are customers of Gigwalk. This suggests that the collection of location-based and time-sensitive data using MCS is a practice of growing importance.

1. https://archive.org/details/201309_foursquare_dataset_umn
2. https://snap.stanford.edu/data/email-Enron.html
3. http://www.gigwalk.com/

- *R. Estrada is with the Escuela Superior Politécnica del Litoral, ESPOL, FIEC & ReDIT Research Group, Guayaquil, Ecuador and with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montréal, QC, Canada. E-mail: restrada@espol.edu.ec.*
- *R. Mizouni is with the Department of ECE, Khalifa University, Abu Dhabi 127788, UAE. E-mail: rabeb.mizouni@kustar.ac.ae.*
- *H. Otrok is with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montréal, QC H3G 1M8, Canada, and with the Department of ECE, Khalifa University, Abu Dhabi 127788, UAE. E-mail: Hadi.Otrok@kustar.ac.ae.*
- *A. Ouali is with the Etisalat British Telecom Innovation Center (EBTIC), Khalifa University, Abu Dhabi 127788, UAE. E-mail: anis.ouali@kustar.ac.ae.*
- *J. Bentahar is with the Concordia Institute for Information Systems Engineering (CIISE), Concordia University, Montréal, QC, Canada. E-mail: bentahar@ciise.concordia.ca.*

Manuscript received 30 May 2016; revised 13 May 2017; accepted 27 June 2017. Date of publication 11 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Rebeca Leonor Estrada.)
Digital Object Identifier no. 10.1109/TSC.2017.2725835

In MCS systems, selection of participants is one of the main challenges, which has an impact on the quality of the task outcome. Several approaches have been proposed to tackle this problem. They aim at selecting the best set of users to complete a task subject to several constraints (e.g., budget). However, most of these approaches are single-task oriented and do not consider the impact of the task allocation problem in a large-scale scenario. There are few solutions that address the multi-task allocation problem (e.g., TaskMe [7] and ActiveCrowd [8]). In these approaches, the relationship between the number of active participants and the number of tasks to be completed affects severely the task allocation and completion rates. Moreover, the multi-task allocation problem faces other challenges such as location dependency, diversity of quality of the sensing data, and budget constraints.

The limitations of the existing approaches are summarized as follows:

- The selection of participants is based on a single-objective optimization problem (e.g., maximizing the number of accomplished tasks, minimizing the budget [9]) assuming that the workers are willing to participate regardless how much they expect to earn. Moreover, modeling the sensing location-based task without specifying any time constraints [10], [11] is unrealistic as assumption for many crowd-sensing systems.
- Task allocation and completion rates are severely affected by limited resources (i.e., active workers) [7], [8]. Therefore, complementary components should be investigated in order to enhance the performance of the task allocation model.
- There is a lack of effective delegation mechanisms. A delegation scheme [12] with monetary incentives can be abused because the participants can misbehave. In other words, workers may compete to perform a task and once they are selected (for instance due to their high reputation), they may delegate their work to other workers and still get paid for the task (that other workers performed).
- The inability to complete tasks affects not only the participants' payment but also their reputation. For example, the approach in [13] does not pay the workers if they give wrong answers, which can discourage the workers to stay committed to the MCS system. The model should update the worker's reputation based on their performance and their successful delegation.

In this paper, our solution addresses the trade-off among quality of the sensed data, budget and time constraints for tasks that require the sensed data within a time frame because the information is useless afterwards. The main contribution of this paper is a service computing framework for the allocation management of time-constrained and location-based sensing tasks that consists of:

- a multi-objective task allocation algorithm to deal with worker selection taking into account that the workers establish their minimum wages to perform any task. This algorithm maximizes the aggregated QoI (Quality of Information)/budget ratio while minimizing the response time under scenarios with time and budget limitation, which is implemented using the Particle Swarm Optimization (PSO) technique.
- two types of queuing schemes: (1) a First In First Out (FIFO) queue implemented in the device application that allows participants to be selected to perform consecutive tasks; and (2) a priority queue in the task manager to queue arriving tasks when there is no available resources and their required response time has not expired.
- a delegation mechanism in case the workers cannot finish their allocated task. Workers may recommend a set of workers from their social network to finish their assigned task. To avoid any abuse of the system, this mechanism affects the recommender reputation based on the performance of the delegated workers.
- a systematic evaluation of workers' reputation based on their performance and the incentives/penalties from the delegation mechanism.

To evaluate the MCS system, we use event-driven simulations [14]. In other words, the functioning of the system is simulated as a discrete sequence of events over time where the occurrence of an event triggers the performance of some actions. Examples of these events include task arrival/departure or mobile user arrival/departure. In this paper, we only consider task arrival/departure events leaving the modeling of mobile user arrival/departure events for future work.

For comparison purposes, we use two benchmark models that are based on the proposed framework using different task allocation algorithms found in the literature. The first algorithm aims at maximizing the quality of information per task under budget constraints [15]. It did not consider time-constrained tasks. We implemented it using the PSO technique. The second one is the heuristic algorithm presented in [16]. We modified both algorithms to include the time constraint and budget estimation as the product of the traveled distance and worker payment. Both benchmark models are evaluated using the other components of the proposed solution, namely, the queueing schemes, delegation mechanism and reputation management. A performance comparison is carried out under two scenarios (1) an incremental scenario where the number of tasks is increased by a step of 15 tasks, which allows us to show that the proposed multi-objective task allocation algorithm can enhance the performance of the benchmark models for different number of tasks in a specific time slot; and (2) a realistic scenario, where the task arrival is modeled as a Poisson process and the required response time per task is an exponential random variable. Simulations are conducted using locations from a real dataset[4] where a large-scale scenario is generated.

The remaining of the paper is organized as follows: Section 2 presents an overview of the relevant related work. Section 3 formulates the multi-task allocation problem and discusses the challenges of such a formulation. Section 4 presents our proposed service computing framework and its components. Section 5 describes the two benchmark models used in this paper. Section 6 presents the simulation scenarios and results. Finally, Section 7 concludes the paper.

2 Related Work

Crowdsensing techniques and challenges are analyzed with a focus on resource constraints and data quality issues in

4. https://archive.org/details/201309_foursquare_dataset_umn

TABLE 1
Location-Based Task Management Solutions

Solution	Contribution	Dataset	Limitations
ProMoT [11]	Auction mechanism that maximizes the profit of the platform while providing satisfying rewards to the workers	Randomly Generated	- No time-constraints - Single-task oriented - Does not take into account workers' reputation
QOATA [15]	Single objective optimization approach to maximize the task QoI with budget constraint taking into account the workers reputation	Randomly Generated	- No time-constraints - Single-task oriented - Does not take into account workers' reputation
Budget Task [16]	Heuristic algorithm for single objective optimization problem that maximizes the task QoI taking into account the workers reputation and payment	Workers and tasks from Foursquare [22] Worker's Reputation is randomly generated	- No time-constraints - Single-task oriented - Payment does not depend on traveled distance
TaskMe [7]	Two bi-objective optimization approaches for participant selection FPMT: to maximize the total number of accomplished tasks and also to minimize the total movement distance. MPFT: to minimize total incentive payments for participants and minimize traveling distance to complete tasks	For workers D4D[23]	- Complexity - Fairness among workers - Does not guarantee the task QoI - Does not take into account workers' reputation - Not suitable for large-scale scenario - Task location are randomly generated within the mobile user area
ActiveCrowd [8]	Two greedy-enhanced genetic algorithms for optimal task allocation to minimize the total distance traveled to complete the tasks under two common situations: 1) intentional-movement-based selection for time-sensitive tasks and 2) unintentional-movement-based selection for delay-tolerant tasks	For workers and tasks D4D [23]	- Not an optimal solution - Does not guarantee the task QoI - Worker Payment is not considered - Complexity - Task/worker locations given by cell towers' location

[17]. A better understanding of resource management and QoS estimation in mobile crowdsensing can help researchers design cost-effective crowdsensing systems that can reduce the cost by fully utilizing the resource and improve the QoI for customers. Regarding task allocation and participant selection problems, the majority of existing solutions are single-task oriented. These approaches do not address the task allocation problem for a large-scale scenario where multiple heterogeneous tasks can be requested by several customers and be performed by several workers. Moreover, the participants selection procedure is based on a single optimization objective (e.g., sensing costs [9], coverage of targets of interest [18], quality or credibility of sensed data [15], [16], or revenue [10], [11]). There are few solutions that address the multi-task allocation problem taking into account several optimization objectives to find a trade-off between the most commonly used factors [7], [8].

Some researchers introduce redundancy to ensure a certain level of reliability in MCS systems [19] and several workers are asked to carry out the same task. Then, a technique such as majority voting [20] is applied to determine the answer for the requester. Although this solution reduces the impact of wrong answers on the final result [21], it increases the required budget to perform a given task. A trade-off between maximizing the aggregated quality of information per task while minimizing the budget per task should be investigated.

Furthermore, once the workers are selected to perform a task, any worker selected may not complete the task due to unforeseen circumstances. If the quality of information of the task is not met, then, a re-selection of workers should be carried out. Instead of performing the re-selection procedure, a delegation mechanism was proposed in [12]. Their mechanism allows a worker who cannot finish the task to recommend other worker from her/his social network to perform the task. For time-constrained tasks, the delegation is more complicated because the data should be collected within a certain time interval.

Table 1 summarizes the main contributions and limitations of five relevant approaches found in the literature related to our work.

3 Formulation of Multi-Task Allocation Problem

In this section, the model for multi-task allocation problem is presented. We consider a service computing framework where the service provider publishes the tasks for the workers. Then, each worker chooses some of the published tasks and provides the minimum payment that the worker is willing to receive from a specific range. The range payment depends on the worker reputation. The workers can visualize the tasks that satisfy some basic constraints such as the workers is within the task coverage radius and the requested payment is lower than the maximum payment per task.

3.1 Problem Formulation

The objective of the multi-task allocation problem (MTAP) is to maximize the ratio of the aggregated QoI to the required budget and response time to perform several tasks given a set of available workers. The MCS should determine the assigned tasks for each worker within their requested response time and time constraints. Workers have different reputation levels based on their historical performance in the MCS system. They are also attributed a confidence level, which represents the self-confidence that a worker has

related to the task accomplishment. For example, the battery usage level of the worker's mobile device can be used as the confidence level. In fact, a user with low battery level is less likely to be selected to perform any task. For location-based tasks, the user needs to travel a certain distance to perform the task. As the distance increases, the user needs more time to travel to the task location. This fact delays the data collection and increases the cost of performing the task (i.e., the system needs to reward the user for travelling a long distance).

While there is no definition of information quality that fits every scenario, this concept is often related to the accuracy of the information, completeness, and timeliness. Many formulations have been proposed in the literature [15], [16]. In our case, the quality of information of a worker c_j^i reflects the accuracy and timeliness of the collected data and is given by

$$c_j^i = r_j \times \beta_j \times \delta_j^i, \tag{1}$$

where r_j and β_j are the reputation and confidence of the worker to perform a given task during a given period of time. The reputation is a parameter computed by the MCS system based on the historical performance of the worker. The confidence β_j is an input parameter reflecting the worker's self-confidence to perform the task. The battery level is an example of such a parameter that could also be expressed by a combination of different other parameters. δ_j^i is a function of the distance between a worker and a given task. We use the same function given in [16], which calculates the discount to the worker's reputation as a result of his proximity to the task location

$$\delta_j^i = 1 - \max\Big(0, \min\Big[log_{DC}\Big(d_j^i\Big), 1\Big]\Big), \tag{2}$$

where DC is the city radius (i.e., 30 km) and d_j^i represents the euclidean distance between the worker location l_j and the task location l^i, which are given in GPS coordinates in the Foursquare dataset. This distance is estimated using the Haversine formula [24].

The objective function for the MTAP problem aims at maximizing the aggregated quality of information per unit of required budget and it can be formulated as

$$\max_{\mathbf{X},\mathbf{P}} \sum_{i \in T} \left[\frac{\Big(\sum_{j \in W} c_j^i X_j^i\Big) - C^i}{\Big(\sum_{j \in W} d_j^i P_j^i\Big) \times \max_{j \in W}(t_j^i)} \right], \tag{3}$$

where $\mathbf{X}$ and $\mathbf{P}$ correspond to the vectors of the variables X_j^i and P_j^i respectively. X_j^i is a binary variable that indicates the selection of a worker j to perform the task i while P_j^i corresponds to the payment per traveled kilometer received by worker j to perform the task i. C^i represents the minimum QoI required by task i. Thus, the numerator represents the aggregated quality of information per task i and the denominator is the product between the required budget and the required time to finish the task i for the selected set of workers given by vector $\mathbf{X}$. W and T are the set of workers and tasks respectively. t_j^i is the estimated time that the worker takes to reach the location of the task τ_i. This time is a calculated as the distance between the worker j and task i divided by the speed of the worker j. The sensing time is assumed to be negligible in comparison to this time.

TABLE 2
Model Parameters

Name	Description
Task Parameters	
T	Set of tasks
B^i	Maximum budget per task i
R^i	Coverage radius of task i
C^i	Minimum QoI for task i
P_{max}^i	Maximum payment allowed per traveled km for workers for task i
$P_{g,max}^i$	Maximum payment per traveled km for worker with reputation level g for task i
$P_{g,min}^i$	Minimum payment per traveled km for worker with reputation level g
DC	Radius of the city
l^i	Location of the task i
N_{max}^i	Maximum number of workers per task i
t_{max}^i	Maximum response time required by Task i
t_q^i	Queuing time for task i
W^i	Set of selected workers to perform task i
S^i	Subset of arbitrary workers in W^i that can perform task i
Worker Parameters	
r_j	reputation of the worker j
$r_j^{(k)}$	reputation of the worker j at the instant k
N_j^{max}	Maximum number of consecutive tasks per worker
W	Set of workers
c_j^i	QoI provided by the worker j to the task i
β_j	Self-Confidence to perform any task of worker j
l_j	Location of worker j
P_j^{min}	Requested payment that the worker is willing to receive per traveled km
d_j^i	Distance from task i to worker j
t_j^i	Time that the worker takes j to reach the location of task i
W_j^{SN}	Set of recommended workers from the social network of worker j
$W_j^{D,i}$	Set of delegated workers to perform the task i that worker j could not do it
N_j^T	Number of assigned task to the worker j at the time k
N_j^{TV}	Number of completed tasks with true value for worker j
N_j^{CT}	Number of completed tasks for worker j
N_j^{IT}	Number of incomplete tasks for worker j
General Parameters	
r_g^{min}	Minimum reputation for level g
r_g^{max}	Maximum reputation for level g
Output Variables	
X_j^i	Binary variable that indicates if task i is allocated to worker j
P_j^i	Value paid per traveled km to the worker j for performing the task i

3.1.1 Model Parameters

For the sake of clarity, Table 2 summarizes the notation used in this paper.

3.1.2 Model Constraints

The objective function (3) is subject to the following constraints

$$\sum_{j \in W} X_j^i \leq N_{max}^i, \qquad i \in T \tag{4}$$

$$\sum_{j \in W} c_j^i X_j^i \geq C^i, \qquad i \in T \tag{5}$$

$$t_j^i \times X_j^i \leq t_{max}^i, \qquad ; i \in T, j \in W \tag{6}$$

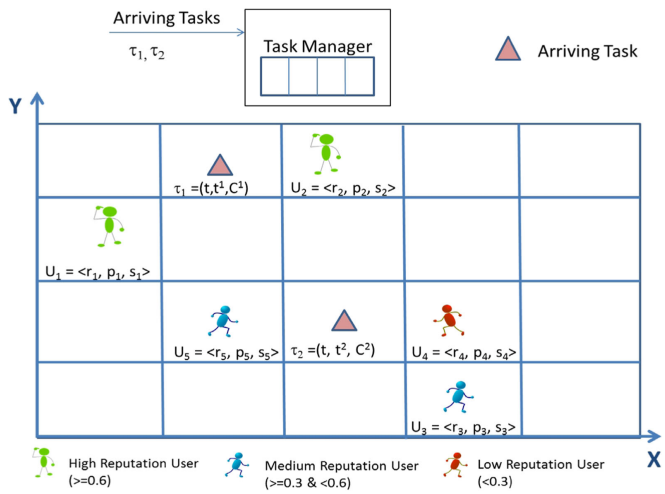


Fig. 1. Scenario with two tasks and five mobile users.

$$P_j^i \leq X_j^i P_{g,max}^i, \qquad ;i \in T, j \in W \tag{7}$$

$$P_j^i \geq X_j^i \max(P_j^{min}, P_{g,min}^i) \qquad ;i \in T, j \in W. \tag{8}$$

Constraint (4) defines the maximum number of workers that can be allocated to perform a task i. Constraints (5) and (6) ensure that the set of selected workers will satisfy the quality of information required by task i and deliver the sensed data within the required response time. Finally, constraints (7) and (8) determine the upper and lower bound for the payment of the workers with reputation g.

This model aims at maximizing the objective function (3) and is a non-linear mixed integer problem (MINLP). It could be solved by decomposing the complex problem into several subproblems for each task $i \in T$ [15] or several subproblems for each worker $j \in W$ [25].

3.2 NP-Hardness

Theorem 1. *The MTAP as shown in Eqs. (3), (4), (5), (6), (7), (8) is NP-hard.*

Proof. We prove this theorem by showing that an arbitrary instance of a NP-complete problem can be polynomially Turing reduced to an instance of the MTAP′, a simplified version of the MTAP where each worker has to be assigned to only one task. Thus, the idea is to provide an algorithm that solves the NP-complete problem in polynomial time by calling an oracle that solves the MTAP′. The candidate NP-complete problem we consider is the two-way Number Partitioning Problem (2-NPP) [26].

Given a multiset S of natural numbers, the 2-NPP consists of partitioning S into two subsets S_1 and S_2, so that the sum of the numbers in S_1 is nearly equal to the sum of the numbers in S_2. This problem can be simply solved by calling an oracle to the MTAP′ where two sensing tasks T_1 and T_2 localized in the same region l_1 have to be assigned to a set of workers sharing the same location l_2 so that d_j^i are the same for all the pairs of workers j and sensing tasks i. The total number of workers is equal to $|S|$. All the workers are paid the same price per traveled kilometer so that P_j^i are equal for all the pairs i and j. Moreover, the workers have the same speed s_j of movement so that t_j^i are also equal for all the pairs. Each number $s \in S$ is associated to a worker j who is characterized by two indistinguishable qualities c_j^1 and c_j^2 (i.e., $s = c_j^1 = c_j^2$). These numbers are linked to C^1 and C^2 as follows: $\sum_{s \in S} s = \sum_{j \in W} c_j^1 = \sum_{j \in W} c_j^2 = C^1 + C^2$.

If $\sum_{j \in W} c_j^1$ is even, then set $C^1 = C^2$, otherwise, set $C^1 = C^2 + 1$. N_{max}^1 and N_{max}^2 are set to be large enough so that constraint (4) is always satisfied. If the oracle call provides a solution, then two subsets of workers are identified so that the sum of qualities c_j^i is getting maximized, and according to constraint (5), the sum of these qualities in the two sets are nearly equal. This provides a solution to the 2-NPP since the numbers in the two sets S_1 and S_2 are mapped to c_j^i. If the call to the oracle does not provide a solution, then the oracle is called again after updating C^1 and C^2 as follows: $C^1 := C^1 + 1$ and $C^2 := C^2 - 1$. This procedure is repeated until a solution is provided. The procedure is guaranteed to terminate since the only reason of not providing a solution is the non-satisfaction of constraint (5), and each iteration is modifying C^1 and C^2 towards the satisfaction of the inequality.

The proposed procedure runs in $\mathcal{O}(\sum_{j \in W} c_j^1)$ as there are at most $C^1 + C^2$ calls to the oracle. Therefore, the reduction is polynomial since constructing the 2-NPP solution is a simple mapping of the quantities c_j^i of the workers conducting task t_i to the subset S_i. Thus, the NP-hardness of MTAP′ follows from the fact that 2-NPP $\preceq_p$ MTAP′, where $\preceq_p$ is the polynomial Turing reduction. Since the MTAP is harder than the MTAP′ because in the MTAP a worker can be assigned to 0 or many tasks, which increases the number of possible combinations, we conclude that the MTAP is NP-hard. □

TABLE 3
Worker Parameters and Metrics

Worker	r_j	P_j^{min}	s_j	d_j^1	d_j^2	t_j^1	t_j^2	c_j^1	c_j^2
U_1	0.85	5	1	1.41	2.2	1.41	2.2	0.76	0.65
U_2	0.8	4.2	0.75	1	2	1.33	2.67	0.8	0.64
U_3	0.5	2.5	0.25	3.61	1.41	14.44	5.64	0.31	0.45
U_4	0.25	0.5	1	2.82	1	2.82	1	0.17	0.25
U_5	0.55	2.7	0.5	2	1	4	2	0.44	0.55

3.3 Motivating Example

Let's consider a simple MCS system with five available workers $(U_1, .., U_5)$ and two arriving tasks (τ_1, τ_2) in a two dimensional (2D) area as illustrated in Fig. 1. This simplified example is only for illustrative purpose; real scenarios are more complex and highly scalable involving considerable number of tasks and workers. Workers might have different reputations (e.g., high, medium and low reputation users). The question is how to allocate these workers to each task so that they can maximize the aggregated quality of information per unit of used budget and time to execute each task. The required quality of information are 1 and 0.9 for tasks τ_1, and τ_2 respectively while response time is 5 minutes for both tasks.

Table 3 presents the worker parameters, namely reputation, speed, minimum payment to receive as well as the

TABLE 4
Set of Workers to Perform Task τ_1

Workers	QoI	Agg QoI	Budget	Response Time	$\frac{AggQoI}{B\times T}$
$[U_1, U_2]$	1.56	0.56	11.25	1.41	0.04
$[U_1, U_2, U_4]$	1.74	0.74	12.66	2.82	0.02
$[U_1, U_2, U_4, U_5]$	2.18	1.18	18.06	4.00	0.02
$[U_1, U_2, U_5]$	2.00	1	16.65	4.00	0.02
$[U_2, U_4, U_5]$	1.41	0.41	11.01	4.00	0.01
$[U_1, U_4, U_5]$	1.38	0.38	13.86	4.00	0.01
$[U_2, U_5]$	1.24	0.24	9.60	4.00	0.01

estimated distance to reach the task locations and the estimated quality of information that each worker can contribute to each task using Eq. (1). We include a minimum payment per worker to represent the willingness of the worker to perform a task. This minimum payment depends on the corresponding range of the worker reputation $\langle P^i_{g,min}, P^i_{g,max}\rangle$. We assume that the range payment for each reputation level are the same for both tasks. For instance, high reputation users can select values in the range between 3.5 and 5 while medium and low reputation users can select values from [2, 3.5) and [0.5, 2) respectively. For the motivating example, we assume that the worker confidence is equal to 1 for any task.

Table 4 presents the worker combinations that meet the requirements for task τ_1. We start with task τ_1 because it requires higher QoI. Then, the problem is to select disjoint set of mobile users to solve each task since they cannot perform both tasks at the same time.

Selecting the first option for task τ_1 could mean that only three workers can be allocated to task τ_2. Then, only one combination of these workers meet the QoI of task τ_2, which is shown in the scenario I in Table 5. If the model only maximizes the QoI per task under budget constraint, then, this combination could be selected as a solution. However, this implies that the worker with higher response time do not get paid and the worker satisfaction is decreased.

In the second scenario of Table 5, the task manager allows each user to keep a local queue of tasks to be carried out sequentially. Thus, the users U_1 and U_2 that were allocated to perform task τ_1, can also perform task τ_2 after finishing task τ_1. This scenario presents the worker combinations including users U_1 and U_2 taking into consideration the total time that these users need to reach the location of task τ_2 while their payments are estimated using the traveled distance from the location of task τ_1 to the location of task τ_2. In this case, the first option is the one that maximizes the ratio between aggregated QoI divided by the product of budget and response time and also guarantee the worker payment. This means that workers U_1 and U_2 will perform two consecutive tasks while workers U_4 and U_5 will carry out one task. In addition, all the workers performing tasks will receive their payment owing to the fact that they can deliver the sensed information within the requested time. Worker U_3 is not selected because the sensed data cannot be delivered to the task manager within the required response time. This allows the MCS system to meet the requirements of task τ_2 and to enhance the worker satisfaction.

In summary, the MTAP model proposed assigns all tasks to the available workers in one step. We demonstrated by means of an example that this might not be the best strategy and it would be better if some workers are allowed to execute several tasks sequentially. The following section presents an enhanced task manager framework that deals with the single task allocation at a time and allows the workers to be selected to perform more than one task. The workers are considered for the allocation of new tasks as long as they can reach the new task location within its required time and they do not reach their maximum number of tasks allowed by the system.

TABLE 5
Set of Workers to Perform Task τ_2 under Two Scenarios

Scenario I: U_1 and U_2 are not included					
Workers	QoI	Agg QoI	Budget	Response Time	$\frac{AggQoI}{B\times T}$
$[U_3, U_4, U_5]$	0.92	0.02	6.73	5.64	0.0006
Scenario II: U_1 and U_2 are included					
Workers	QoI	Agg QoI	Budget	Response Time	$\frac{AggQoI}{B\times T}$
$[U_1, U_2, U_4, U_5]$	2.06	1.16	23.44	4.26	0.012
$[U_1, U_4, U_5]$	1.45	0.55	14.2	3.61	0.011
$[U_2, U_4, U_5]$	1.41	0.51	12.44	4.26	0.009
$[U_1, U_2, U_4]$	1.51	0.61	20.74	4.26	0.006
$[U_1, U_5]$	1.2	0.3	13.7	3.6	0.006
$[U_2, U_5]$	1.16	0.26	11.94	4.26	0.005

4 Service Computing Framework for Task Management in MCS System

In this section, we present the proposed framework for task management in the MCS system. The main idea is that the MCS task manager can be considered as a queuing system with N servers (mobile users), which can process the same task in parallel and at the same time several tasks can be allocated for service if there are some available servers. Thus, our MCS system can be seen as discrete-event system where the system state $S(t)$ is defined by the number of tasks in service and the number of queued tasks $(T_s(t), T_q(t))$. The task arrival process is a Poisson process while the service time per server is deterministic.

4.1 Multi-Objective Task Allocation Model

Here, we present the optimization problem for the worker selection per task that maximizes the ratio between the aggregated quality of information, the product of the budget,and the execution time under the constraints to meet the quality of information with the limited budget and response time. This means that our objective function for every task i is

$$\max_{\mathbf{X},\mathbf{P}} \frac{\left(\sum_{j\in W} c^i_j X^i_j\right) - C^i}{\left(\sum_{j\in W} d^i_j P^i_j\right) \times \max_{j\in W}(t^i_j)}. \tag{9}$$

The denominator in (9) corresponds to the task budget multiplied by the response time to gather the information from the allocated workers to the task (i.e., the maximum time of the allocated workers to perform a task). The budget is estimated as the payment paid per traveled kilometer to the worker, P^i_j, multiplied by the traveled distance per worker, d^i_j. The second term in the denominator is introduced to reduce the total required time to collect the sensed information.

In our model, a spatial task i is represented as a tuple of the form $\langle l^i; C^i; R^i; B^i; P^i_{max}; t^i_{max}\rangle$. These parameters corresponds to the task location, minimum expected QoI, coverage radius, budget, maximum payment per traveled kilometer, and maximum response time respectively. Workers are represented by tuples of the form $\langle l_j; r_j; \beta_j; P^{min}_j; s_j\rangle$, which represent their location, reputation, confidence to perform a task, minimum payment they are willing to receive and finally their speed.

In summary, we want to maximize the aggregated QoI/budget ratio while minimizing the time to collect the information about a given task from the workers taking into account the worker's willingness to perform this task. Our objective function is a multi-objective function. Particle Swarm Optimization has been used to solve several complex optimization problems with multi-objective function. Moreover, PSO has been proven to obtain a satisfying solution while speeding up the optimization process in comparison to other evolutionary-based optimization algorithms [27]. Therefore, we propose to solve the optimization problem using this technique.

4.1.1 Model Constraints

The constraints for our task allocation sub-problem are:

- Maximum Budget per task i

$$\sum_{j\in W} P^i_j d^i_j \leq B^i. \tag{10}$$

- Minimum required Quality of Information

$$\sum_{j\in W} c^i_j X^i_j \geq C^i. \tag{11}$$

- Maximum response time

$$t^i_j \times X^i_j \leq (t^i_{max} - t^i_q), \tag{12}$$

and the constraints (7) and (8) given for MTAP model.

4.1.2 PSO-Based Multi-Objective Task Allocation Algorithm (PSO-MOA)

We propose to solve the worker selection for each task defined by Eq. (9) using PSO, which is a population-based search approach and depends on information sharing among the population members to enhance the search processes using a combination of deterministic and probabilistic rules. PSO algorithm uses two vectors that determine the position and velocity of each particle n at each iteration k. These two vectors are updated based on the memory gained by each particle. The position y^k_n and velocity v^k_n of a particle n at each iteration k are updated as follows:

$$y^k_n = y^{k-1}_n + \delta_t v^{k-1}_n, \tag{13}$$

$$v^k_n = \omega v^{k-1}_n + c_1 r_1 (p^{local}_{k-1} - y^{k-1}_n) + c_2 r_2 (p^{global}_{k-1} - y^{k-1}_n), \tag{14}$$

where δ_t is the time step value typically considered as unit [28], p^{local}_{k-1} and p^{global}_{k-1} are the best ever position of particle n and the best global position of the entire swarm so far, and r_1 and r_2 represent random numbers from interval [0,1]. The parameters ω, c_1 and c_2 are the configuration parameters that determine the PSO convergence. The first term is related to the particle inertia ω, which is used to control the exploration abilities of the swarm. Large inertia values produce higher velocity updates allowing the algorithm to explore the search space globally. Conversely, small inertia values force the velocity to concentrate in a local region of the search space. Parameters c_1 and c_2 are known as the cognitive scaling and social scaling factors. Thus, the second and third terms are associated with cognitive knowledge that each "particle" has experienced and the social interactions among "particles" in the population respectively [29].

Many PSO variants update the inertia parameter ω using different functions [29]. For simplicity, we consider the following

$$\omega^n_k = \omega_o \times \frac{(1 - \sqrt{(p^{local}_k - y^k_n)^2 + (p^{global}_k - y^k_n)^2})}{\max(\sqrt{p^{local}_k - y^k_n)^2 + (p^{global}_k - y^k_n)^2})}. \tag{15}$$

According to [28], the convergence of PSO is guaranteed if the following set of stability conditions are met

$$0 \leq (c_1 + c_2) \leq 4 \text{ and } \frac{c_1 + c_2}{2} - 1 \leq \omega \leq 1.$$

For our PSO-based multi-objective task allocation algorithm, the position particle **Y** in the search space is given by two vectors **(X, P)**, which represent the allocation of the task i to worker j and price per worker respectively.

PSO algorithm is formulated as an unconstrained optimizer. One way to accommodate constraints is to augment the objective function with penalties proportional to the degree of constraint infeasibility. In our PSO algorithm, a penalty parameter-less scheme [30] is used to accommodate the constraints, where the penalties are based on the average of the objective function and the level of violation of each constraint during each iteration. According to [28], the penalty coefficients pc_l are determined by

$$pc_l = |\overline{f}(y)| \frac{\overline{g_l}(y)}{\sum_{j=1}^{PC} [\overline{g}(y)]^2}, \tag{16}$$

where l indicates a particular constraint, $\overline{f}(y)$ is the average objective function, $\overline{g}(y)$ is the average level of l_{th} constraint violation over the current population and PC is the number of penalty coefficients, which also corresponds to the total number of constraints [28]. Thus, the fitness function is defined by

$$f'(y) = \begin{cases} f(y^k_n), & \text{if } y^k_n \text{ is feasible} \\ f(y^k_n) + \sum_{l=1}^{PC} pc_l \widehat{g}(y^k_n), & \text{otherwise.} \end{cases} \tag{17}$$

and $\widehat{g}(y^k_n)$ is determined as follows:

$$\widehat{g}(y^k_n) = \max\big(0, g_j(y^k_n)\big). \tag{18}$$

Accordingly, the average of the fitness function for any population is approximately equal to $\overline{f}(y) + |\overline{f}(y)|$. Since we formulate our model as a maximization problem and PSO is defined to solve a minimization problem, we modify our objective function from (9) to

$$f(\mathbf{X}, P) = Z - \left(\frac{\left(\sum_{j \in W} c_j^i X_j^i\right) - C^i}{\left(\sum_{j \in W} d_j^i P_j^i\right) \times \max_j(t_j)} \right), \quad (19)$$

where Z is a large number, which is computed as the objective function (9) under the worst case scenario with the minimum response time and maximum budget that a task can allow. The resulting values is then multiplied by 100 to ensure Z to be large enough. The fitness function of our minimization problem is given by

$$f'(x) = \begin{cases} f(\mathbf{X}, P), & \text{for feasible solutions} \\ f(\mathbf{X}, P) + \sum_{l=1}^{PC} pc_l \times \widehat{g}(X, P), & \text{otherwise} \end{cases}. \quad (20)$$

where model constraints are included in $\sum_{l=1}^{PC} pc_l \widehat{g}(X, P)$ to penalize unfeasible solutions. Algorithm 1 presents the PSO-based multi-objective task allocation algorithm. The random function in the Algorithm 1 returns a random number between 0 and 1.

Algorithm 1. PSO-MOA Algorithm

Data: Worker Locations (l_j),
Worker Demands (P_j^{min}),
Task Location (l^i),
Maximum Budget per Task B^i,
Task Maximum Price P_{max}^i
Coverage radius d^i
Required Time t^i)
Result: Set of worker allocated to the task and the price to be paid per worker (X_j^i, P_j^i).
begin
Generate initial swarm with the particle positions $Y_j^i = (X_j^i, P_j^i)$ and velocities randomly v_j^i;
Evaluate Fitness Function;
Determine first global best of the swarm;
while $k \leq MaxIteration$ **do**
Update Position using Eq. (13);
Evaluate Fitness Function;
Determine best local for each particle;
Determine best global in the swarm and update the best global;
Update the inertia parameter w using Eq. (15);
Update velocity using Eq. (14);
end
end

PSO Parameters Settings and Convergence Analysis. The convergence analysis of the proposed PSO algorithm using different values of cognition and social behavior factors (c_1, c_2) is shown in Fig. 2. It can be observed that the best objective value was given for the setting c_1=2 and c_2 = 1.5 after 400 iterations. Therefore, we set the parameters to those values for the rest of our simulations.

4.2 Queuing Schemes

4.2.1 Task Queuing in the Device Application

Our framework proposes to have a local queue in the worker's device with a maximum number of consecutive tasks that can be allocated to him, N_j^{max}. Thus, several tasks can be assigned to a worker and they are going to be performed

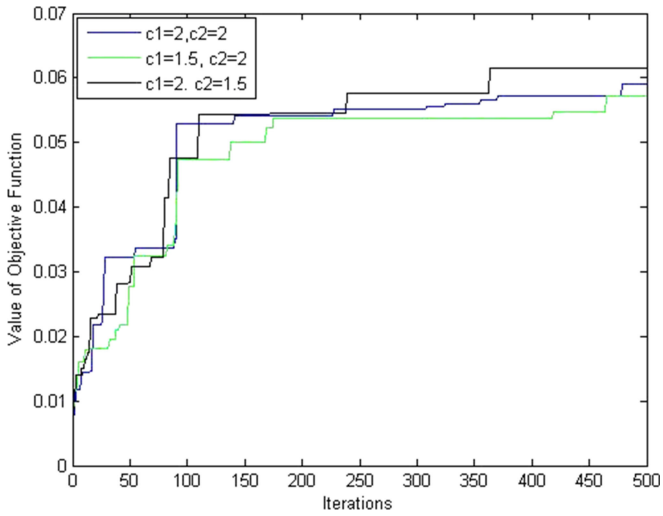


Fig. 2. Convergence Analysis for different settings of c_1, c_2.

in the sequential order that they were assigned to the worker (i.e., FIFO queue). The main idea is to reduce the number of rejected tasks in a scenario with reduced number of workers within the task coverage area. Thus, the following constraint is added to our task allocation model

$$X_j^i + N_j^T \leq N_j^{max} \qquad ; j \in W, \quad (21)$$

where N_j^T is the number of assigned task to the worker j at the time k without being processed and N_j^{max} is the size of the worker local queue. For convenience, we assume that all the workers have a local queue with the same size, (i.e., $N_j^{max} = N_w^T; \quad \forall j \in W$).

4.2.2 Priority-Based Task Queuing in Task Manager

Our framework also uses a priority-based queuing system in the task manager. If there are no available workers to perform the task, then, the task is queued until the potential workers are released from their current assigned task (i.e., after finishing their assigned tasks) and can perform the queued task. Otherwise, the task is rejected. The priority is defined by the remaining response time. Thus, a low remaining response time task should be assigned first over the tasks with higher time. Doing so, the model always selects the task that needs to be served first according to the remaining response time. If the remaining response time is zero, then, the task is eliminated from the queue and it is marked as an unsuccessful queued task. The task allocation algorithm complements the queuing scheme since it aims at minimizing the response time per task. This means that the workers are able to finish rapidly their current tasks and they can be assigned to the new arriving or queued tasks.

4.3 Task Delegation Mechanism

The proposed task delegation mechanism is intended to avoid the QoI decrease when some workers are not able to finish their assigned task(s) due to unpredictable circumstances. In our proposal, a worker who cannot finish a task is able to recommend one or a worker set W_j^{SN} from the social network to perform the task. Since the worker's social network may be unknown to the MCS system, the task manager determines the appropriate set W_j^D from the set of

recommended workers W_j^{SN}, who are registered in the system and can satisfy the following requirements:

- The sum of the QoI of selected workers should be at least equal to QoI of the recommender.

$$\sum_{h \in W_j^{D,i}} c_h^i \geq c_j^i, \qquad ; i \in T, \tag{22}$$

where $W_j^{D,i}$ represents the selected delegated workers for task i obtained from the set of recommended workers W_j^{SN} by the worker j. This means that $W_j^{D,i}$ is a subset of the intersection of the set of workers registered to MCS system and the set of delegated workers from the social network of worker j ($W_j^{D,i} \subseteq W_j^{D,i} \cap W$) that comply with the requirements of task i. The index h is used to represent workers other than j.

- The maximum budget to be allocated to the delegated workers should be less or equal to the remaining budget.

$$\sum_{h \in W_j^{D,i}} P_h^i d_h^i \leq B^i - \sum_{k \in W^i \setminus j} P_k^i d_k^i, \tag{23}$$

where W^i is the set of workers selected to perform the task i by the task manager.

- When delegation is allowed, the delegated workers may be located outside the area of coverage. However, they should reach the task location within the remaining response time. This constraint is given by

$$t_h^i \leq t_{max}^i - \max_{k \in W^i} t_k^i. \tag{24}$$

- The probability of not finishing a task for the delegated workers is exponentially reduced (e.g., p^2).
- Only one delegation level is taken into account, which means delegated workers are not allowed to delegate.
- The model incentivizes the recommender by increasing their reputation as linear function of the reputation of the delegated workers. Doing so, the user avoids getting bad reputation for future tasks and the delegation becomes useful for time-constrained tasks.

4.4 Reputation Management

The worker reputation should be updated every time that a given number of tasks are completed in the system. In this procedure, only the workers that have been allocated to tasks are considered regardless of the task completion or delegation. Our framework updates the worker reputation each time that five tasks are completed. The reputation of the participating worker j at time k is estimated as follows:

$$r_j^k = min\left(1, \frac{N_j^{TV}}{N_j^{CT} + N_j^{IT}} + \sum_{l \in W_j^{D_{k,k-1}}} \alpha * r_l^{k-1}\right), \tag{25}$$

where the first term corresponds to the worker performance evaluation in the MCS system. This means how well is the worker performing his assigned tasks. N_j^{TV}, N_j^{CT} and N_j^{IT} indicates the number of completed tasks with true value, number of completed tasks and number of incomplete task respectively for the worker j. The proposed framework considers that a task is answered with a true value by the workers if their answers are equal to the estimated ground truth value by the system, which is given in (26). The second term is related to the delegation mechanism and $W_j^{D_{k,k-1}}$ corresponds to the set of delegated workers by the worker j during the reputation update periods between k and $k-1$. This term is a linear function of the a given parameter α multiplied by the reputation of the delegated workers in the previous period. The value of α can be positive or negative and depends on the task completion of the delegated worker. If the delegated worker did not finish the task this parameter is negative, otherwise it is positive. r_l^{k-1} represents the reputation of worker l in the previous period $k-1$.

We assume that the ground truth of the task (i.e., the correct answer to the location-based sensing task) is binary as in [16], which is reasonable for many real-world situations (e.g., whether the road work at a particular location has been completed, on-shelf availability of a product in a convenience store, etc). In practice, the correct answer of a task i is unknown to the task requester which makes the performance evaluation of the participating workers difficult. To overcome this limitation, the authors in [16] propose to estimate the ground truth from the collected data o_j^i using the majority voting system [20]. This means that the ground truth is the most common response (vote) given by the selected workers (voters) regardless the worker's reputation. Our MCS system uses instead the weighted voting system [31] based on the idea that not all workers have the same influence over the estimated ground truth. In other words, workers with high reputation are more reliable to give good answers than the ones with low reputation. Thus, the estimated ground truth O^i of the task i is given by

$$O^i = \lfloor \frac{\sum_{j \in W^i} X_j^i r_j o_j^i}{\sum_{j \in W^i} X_j^i r_j} - \frac{1}{2} \rfloor + 1. \tag{26}$$

In other words, O^i = 1 if the average outcome is greater than $\frac{1}{2}$, and is 0 otherwise. This mechanism allows the model to evaluate the performance of the participating workers. Thus, if the worker response o_j^i is equal to the estimated ground truth O^i. If so, it is assumed that the worker completed the task i with a true value. We keep the historical worker performance for the allocated tasks to update their reputation over time. However, if the ground truth of the sensing tasks are not binaries, it would be good to use other mechanism to estimate the ground truth of the tasks. Regardless the mechanism used to determine the ground truth, the MCS framework needs to know if the reported value of the worker is close to the ground truth of a task. If so, it is assumed that the worker completed the task successfully.

4.5 Running Example

Based on the motivating example presented in Section 3.3, let's suppose now that two new tasks (τ_3, τ_4) arrive to the system one minute after the arrival of task τ_1. The minimum quality of information for both tasks is 1 and the maximum response time is 10 and 5 respectively. In this example, we limit the maximum number of tasks per worker to 2. After 1 minute, the conditions of the workers are shown in Fig. 3.

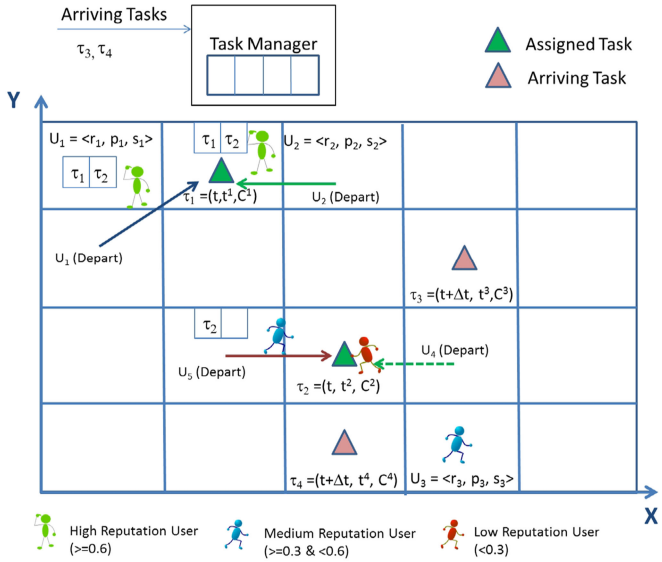


Fig. 3. Scenario with four tasks (2 already assigned and 2 new arriving tasks).

As shown in Table 3, users U_1 and U_2 are still going to the location of their first assigned task (i.e., τ_1) and keep the two tasks in their local queue, which means that they cannot be considered for the allocation task process of the new tasks. Table 6 shows the distances, required time and QoI for each worker regarding the new tasks.

Although both tasks have the same QoI requirement, the algorithm selects task τ_4 to allocate the workers first because of its lowest response time (5 minutes).

One combination meets the requirements of time and QoI as shown in Table 7. After the allocation of this combination to task τ_4, the allocation of task τ_3 is analyzed with the workers having space in their local queue. Only users U_3 and U_4 can be considered for task τ_3 because worker U_5 queue is full. The algorithm also should consider that user U_4 needs some time to finish task τ_4 and the traveled distance to perform task τ_3 (i.e., starting from the location of task τ_4). After running the algorithm for task τ_3, there is no combination that meet the required QoI. However, since the response time of the task is long enough and there are some potential workers that can be selected in a future time, our framework queues task τ_3 in the task manager queue until any other user can be allocated to task τ_3. For example, user U_2 and U_1 can be allocated to another task after 0.33 and 0.41 minutes from the arrival of task τ_3.

After the worker selection is performed for a given task, there is always a probability p that the worker might not be able to finish his/her assigned task. In such scenario, the task manager should wait until all workers deliver their sensed data to see if the QoI is satisfied. When the QoI is deprived, the task manager checks for the recommendations of the worker who did not finish the task and select the delegated workers that can perform the corresponding task within the remaining response time and budget. If the worker did not recommend anyone, this affects the worker reputation since the task is marked as not completed on his historical performance. Otherwise, the task manager rewards or penalizes the reputation based on the recommended participants' performance.

TABLE 6
Worker Metrics for Tasks τ_3 and τ_4

Worker	d_j^3	d_j^4	t_j^3	t_j^4	c_j^3	c_j^4
U_3	2	1	8	4	0.40	0.5
U_4	1.41	1	1.41	1	0.25	0.25
U_5	1.41	1	3.82	3	0.49	0.55

TABLE 7
Set of Workers to Perform Task τ_4

Workers	QoI	Agg QoI	Budget	Response Time	$\frac{AggQoI}{B \times T}$
$[U_3, U_4, U_5]$	1.3	0.3	5.7	4	0.01

5 Benchmark Models and Performance Metrics

In this section, we present two benchmark models (PSO-QoI and QoI-Heu) as well as the performance metrics used to validate and evaluate the proposed framework. The benchmark models use different task allocation algorithms. Also, we extended them using the other components from our framework, namely, queueing schemes, delegation mechanism and reputation management.

5.1 Benchmark Models

5.1.1 QoI Aware PSO-Based Algorithm (PSO-QoI)

This benchmark model aims at maximizing the total quality of information under budget constraints [15]. The authors did not consider time-constrained tasks. To have a fair comparison, we modified their model to include the time constraints. The objective function for this optimization problem is given by

$$\max_{\mathbf{X},\mathbf{P}} \sum_{j \in W} c_j^i X_j^i. \tag{27}$$

We propose to solve the optimization problem under the same constraints (7-8,10,11,12) as in our model using PSO technique. Therefore, the PSO algorithm is similar to the algorithm in 1 but with a different fitness function.

5.1.2 QoI Aware Heuristic Algorithm with Budget Constraints (QoI-Heu)

In [16], a heuristic algorithm, which aims at maximizing the QoI of one task under budget constraint, was proposed. Thus, we modified their algorithm to solve the problem of allocating a location-based task i to a set of candidate workers W subject to a budget limit of $B^i \in R^+$ within the response time $t_{max}^i \in R^+$. The Algorithm 2 presents the modified version of the algorithm in [16].

This algorithm should provides close results to the PSO-QoI algorithm since both algorithms aim at maximizing the QoI per task.

5.2 Metrics

- *Task allocation rate.* This metric represents the percentage of tasks being effectively allocated to workers

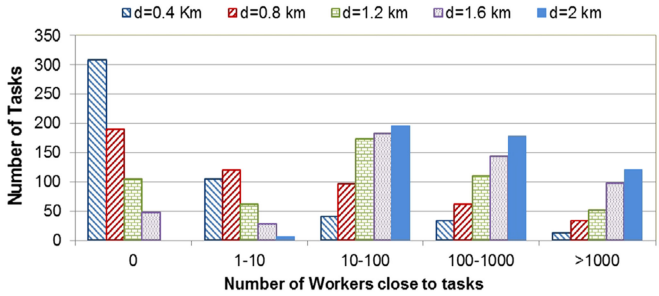


Fig. 4. Range of Workers close to tasks vs Number of task.

$$\overline{\phi_T} = \frac{T_{assigned}}{|T|}, \tag{28}$$

where $T_{assigned}$ is the number of tasks that are effectively allocated and performed within their respective response time.

- *Average Response time per task*. It indicates the average time to perform a location-based task

$$\overline{t_T} = \frac{\sum_{i\in T} \max_{j\in W}(t_j^i)}{T_{assigned}}. \tag{29}$$

- *Average QoI Satisfaction per Task*. This metric measures the average satisfaction of the quality of information over the set of tasks in a given instant

$$\overline{S_{QoI}} = \frac{\sum_{i\in T} max\left(1, \sum_{j\in W} c_j^i X_j^i - C^i\right)}{T_{assigned}}. \tag{30}$$

- *Average Payment per Worker*. It indicates the average payment received by the worker per traveled kilometer and it can be expressed as follows:

$$\overline{P_W} = \frac{\sum_{i\in T}\sum_{j\in W} d_j^i P_j^i}{\sum_{i\in T}\sum_{j\in W} d_j^i X_j^i}. \tag{31}$$

- *Average Estimation Error Rate*. This metric measures how effective is an approach in finding credible workers for a location-based task. The average estimation error rate ϵ is the ratio of incorrect responses (i.e., the number of answers provided by the worker that differ from the ground truth of the assigned tasks) to the total number of assigned tasks. This metric is given by

$$\overline{\epsilon} = \frac{\sum_{i\in T}\left(\frac{\sum_{j\in W} X_j^i \mathbf{1}|_{o_j^i \neq O_i}}{\sum_{j\in W} X_j^i}\right)}{T_{assigned}}. \tag{32}$$

- *Average Budget per Task*. This metric measures the average budget used per task and is given by

$$\overline{B_T} = \frac{\sum_{i\in T}\sum_{j\in W} P_j^i d_j^i}{T_{assigned}}. \tag{33}$$

- *Average Reputation per Worker*. It measures the average reputation of the participating workers in the MCS system

TABLE 8
Workers Distribution per Reputation Level

Number Workers	Initial Distribution		
	Rep_L	Rep_M	Rep_H
200	27	72	101
400	53	141	206
600	70	228	302
800	102	301	397
1,000	118	384	498

$$\overline{R_W} = \frac{\sum_{i\in T}\sum_{j\in W} r_j^k X_j^i}{\sum_{i\in T}\sum_{j\in W} X_j^i}. \tag{34}$$

- *Effective Crowd Size*. This metric measures the number of participating workers in the MCS system

$$\overline{SIZE} = \sum_{i\in T}\sum_{j\in W} X_j^i. \tag{35}$$

6 Simulation Results

For our simulations, we used a real dataset of an existing application: Foursquare. Specifically, two files from this dataset are used: 1) the venues' file that represents the task locations and 2) the users' file that corresponds to the workers' locations. Moreover, we extracted two subsets: 500 venues and 16,836 users to represent the tasks and workers in our model. These subsets allows us to construct realistic spatial crowd sensing scenario to demonstrate that the proposed approach outperforms existing approaches.

Algorithm 2. Heuristic Algorithm (QoI-Heu)

Data: A set of workers W, a spatial task i
Result: Worker selected to the task i and the price to be paid per worker $(X_j^i, P_j^i.)$
begin
  for $i \leftarrow 1$ **to** $|W|$ **do**
    Compute c_j^i according to Eq. (1);
  end
  Rank workers in descending order of their c_j^i ;
  for $j \leftarrow 1$ **to** $min(|W^i|, \lfloor \frac{B}{P_{H,j}^{min}} \rfloor)$ **do**
    $J \leftarrow \min|W^i|, \lfloor \frac{B - P_{H,j}^{max}}{P_{M,j}^{max}} \rfloor$;
    Select a set of workers, W^i who satisfy $d(l_j(t), l_i) \leq R^i$;
    Select a subset,$S^i \in W^i$ workers who satisfy:
    $1 : r_j(t) \geq Th_{ML}$, and;
    $2 : \max_j t_j^i \leq t_i^{req}$;
    (subject to actual availability) with ties broken arbitrarily;
  end
  $P_i = argmax_{S^i} C_{S^i}^i$;
end

We first identify the distribution of the workers in the vicinity of the tasks as the task coverage radius increased from 0.4 to 2 km, which is shown in Fig. 4).

For coverage radius equal to 2 km, it is observed that for 100 tasks it is possible to find between 10-1,000 workers in the vicinity of 100 tasks. Therefore, we select a set of 1,000 workers for our simulations and generate their initial reputation randomly. Table 8 shows the workers distribution per reputation level according to their initial reputation.

TABLE 9
Realistic Scenario Parameters

Name	Description	Value
μ	Required Response time mean	30 min
λ	Task arrival rate	1 Task/minute
N^i_{max}	Maximum number of worker per task i	1 or 5
N^{max}_j	Maximum number of task per worker j	1 or 5
B^i	Maximum Budget per Task	100
P^i_{max}	Maximum Price per Task	5
R^i	Task Coverage Radius	2-5 km
Q	MCS Task Manager Queue Size	0 or 5
r_j	Worker Reputation	0 - 1
β_j	Worker Self-Confidence	0.7 - 1
s_j	Worker Speed	10 - 50 km/h
p	Probability of not finishing a task per worker	0.2

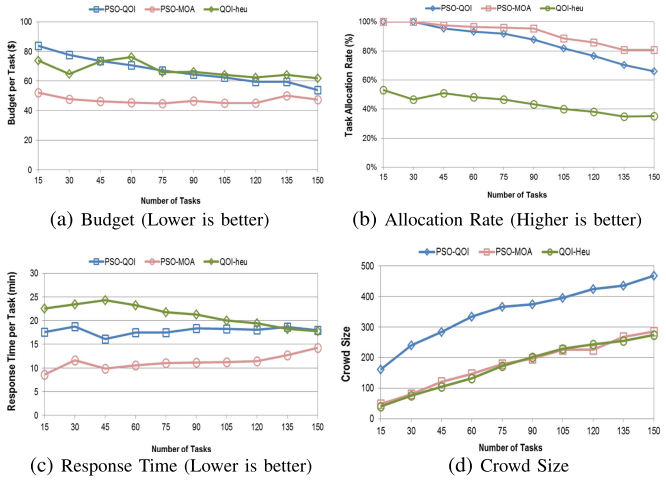
We run the simulations under two different scenarios: an incremental scenario and a realistic scenario.

a) Incremental Scenario. This scenario starts with 15 tasks up to 150 tasks with incremental steps of 15 tasks. This scenario is used to prove that the proposed multi-objective task allocation algorithm can enhance the performance of the benchmark task allocation algorithms for different number of tasks. We modified the three-stage strategy used in [16] to obtain the numerical results. In the first stage, the tasks are sorted according to three parameters: required QoI, the number of workers and requested response time instead of only QoI. The first two parameters are used to sort the tasks in descending order while the response time is used to sort them in ascending order. In such way, the system gives priority to the tasks with short response time. In the second stage, workers are selected if they meet the conditions to perform the task, such as they are located within the coverage area defined by the task and the time to reach the task location is lower than required response time. For this scenario, the performance metrics are presented as a function of the number of tasks in Section 6.1 and the benefits of the worker queue in Section 6.2.1.

b) Realistic scenario. The task arrival process is considered as Poisson Process with parameter λ. Each task identifies the required time to gather the sensed information from the assigned workers. We use an exponential random variable with mean μ to generate the required response time for the tasks. We run extensive event-driven simulations over 1,000 iterations to get to steady state conditions with the parameters described in Table 9.

We assume that workers can finish a task with certain probability $1-p$. If they do not finish a task, their reputation is affected over the time. The initial worker's reputation is randomly generated. Then, after each five completed tasks, the worker reputation is calculated as the ratio between their completed task with correct answer and his total allocated tasks plus the reward or penalty from the delegation mechanism.

The initial worker self-confidence β_j is assumed to be the percentage of his phone battery ($\beta_j \in [0,1]$). In our simulations, the confidence value is a decreasing function of the number of tasks completed by the worker due to the battery consumption needed to perform the task and send the collected data to the server. For convenience, the worker confidence during a given period k is estimated as follows:



(a) Budget (Lower is better) (b) Allocation Rate (Higher is better)
(c) Response Time (Lower is better) (d) Crowd Size

Fig. 5. Performance under an incremental scenario (one task per worker).

$$\beta_j(k) = (\beta_j)^{T^k_j}, \tag{36}$$

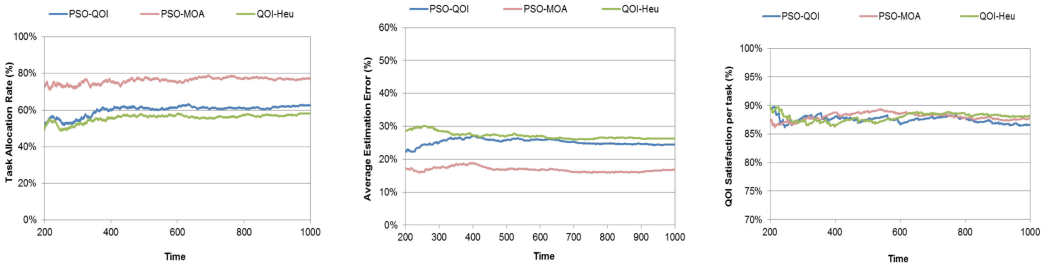
where T^k_j is the number of completed tasks by the worker j during the period k and β_j is the initial measured phone battery level. The battery consumption function is only an example and other functions could be used depending on the technical features and the applications running on the device.

Our event-driven simulation are carried out as follows: At the initial state, one task arrival event is generated. The arrival time follows an exponential distribution. Then, the process starts selecting the event that occurs first (minimum time of occurrence) and according to the type of event (task arrival or departure), several actions are performed.

In the case of arrival, the worker selection for the task starts. If there are available workers, then, the workers that maximize our multi-objective function are selected. Otherwise, the MCS platform queues the task until these workers are available. Once the task is assigned to a set of workers, the type of event is changed to departure and the event time is updated to the service starting time plus the estimated response time from the algorithm and another arrival task event is generated.

In the case of departure, the MCS platform verifies that the selected workers have sent the sensed data. If the QoI of the task is higher than the MCS customer requirements, then, the number of completed tasks is increased by one and the workers' payment and reputation are updated. Otherwise, the MCS platform checks for delegation proposal by the worker who did not finish the task. If the worker did not recommend any other worker, the task is considered as incomplete because the MCS customer won't pay for the sensed data that does not comply with the required quality of information. In the case of delegation, the event time is updated to the total service time estimated by the delegation mechanism. Finally, the MCS platform always checks the queue to allocate the queued tasks as soon as potential workers are available.

For the realistic scenario, the performance metrics over the time are presented in Section 6.1. Then, we show how the delegation mechanism can improve the worker satisfaction without depriving the worker reputation in Section 6.3. In addition, we analyze the impact of each component of



(a) Task Allocation Rate (b) Estimation Error (Lower is better) (c) QoI per Task (Higher is better)

Fig. 6. Performance under a realistic scenario (one task per worker).

the framework individually over the performance metrics in Section 6.4.

6.1 Performance Analysis

Fig. 5 shows the task allocation rate, budget and response time for the three models, where the worker can perform only one task at a time but several workers can be involved in the execution of one task.

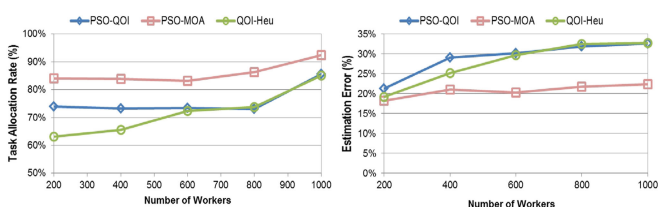
For the case of 15 tasks, PSO-MOA model requires 50 percent less budget than the benchmark models PSO-QoI and QoI-heu and it achieves a task allocation rate of 100 percent. QoI-Heu has a lower task allocation rate (around 55 percent) due to the fact that this model recruits more workers to perform each task, as shown in Fig. 5d. PSO-MOA model also reduces the time to collect the sensed data from the workers by 55 and 40 percent in comparison to the time required by the PSO-QoI and QoI-Heu models respectively.

Fig. 6 depicts the performance metrics for the realistic scenario. In fact, the proposed framework increases the task allocation rate and reduces the estimation error while keeping the same level of QoI satisfaction per task in comparison with the benchmark models. In particular, PSO-MOA model presents a task allocation rate approximately 20 and 25 percent higher than PSO-QoI and QoI-heu models respectively while the estimation error is around 10 percent less than the other two models and the average worker reputation is 10 percent higher than the benchmark models.

In summary, the PSO-MOA model outperforms the benchmark models under the incremental and the realistic scenarios.

TABLE 10
Running Time (sec)

No. Tasks	PSO-QoI	PSO-MOA	QoI-Heu
1	2.1	1.5	0.2
30	28.39	31.47	0.94
60	51.78	60.20	1.78
90	80.22	97.47	3.17



(a) Task Allocation Rate (b) Estimation Error Rate

Fig. 7. Impact of number of available workers.

6.1.1 Complexity

Table 10 presents the running time for the incremental scenario with a number of available workers equal to 1,000. As expected, the running time increases as the number of workers increases. The running time of the three-stage algorithm using PSO for the task allocation is considerably higher than the one using the heuristic algorithm. In the incremental scenario, PSO-QoI algorithm requires an average of 1 minute and 20 seconds to find the worker selection while PSO-MOA requires 1 minute and half approximately for the case of 90 tasks. Nevertheless, it should be noticed from Fig. 5c that the average response time per task plus the running time of the three-stage algorithm is still lower than the response time of the benchmark models. For the realistic scenario, the running time is given by the first row in Table 10 since each task is allocated upon its arrival.

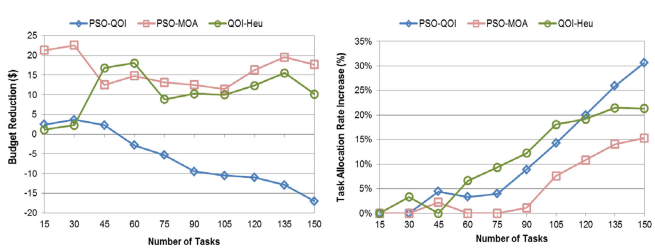
6.1.2 Impact of Crowd Size

Fig. 7 depicts the task allocation rate and the estimation error rate versus the number of available workers for the realistic scenario under steady state conditions.

It can be noticed that our framework presents the highest task allocation rate (85 -92 percent) with the lowest estimation error rate (15 - 22 percent). In particular, QoI-Heu algorithm has similar behavior as the PSO-QoI algorithm when the number of available workers is higher than 600. This means that the QoI-Heu model indeed approximates the results of the PSO-QoI model if the number of available workers is large enough; otherwise, the heuristic algorithm does not perform well.

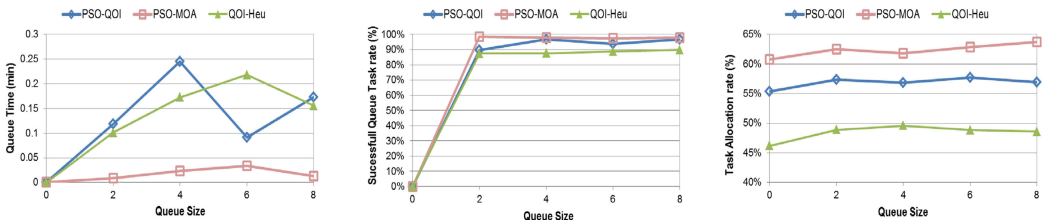
6.2 Queuing Schemes

In the proposed framework, two types of queue are used: a worker queue and the task manager queue as explained in Section 4.2.



(a) Budget Reduction (b) Task Allocation Rate Increase

Fig. 8. Impact of multiple tasks allocation per worker under an incremental scenario (From 1 Task to 5 Tasks per worker).



(a) Queue Time (b) Successful Queued Task Rate (c) Task Allocation Rate

Fig. 9. Impact of size of the priority queue.

6.2.1 Worker Queue

Here, the budget reduction and task allocation rate gain for the incremental scenario is presented in Fig. 8. The budget reduction is the difference between the budgets spent for the cases without and with the local worker queue while the task allocation rate gain is the difference between the task allocation rate for the cases with and without the local worker queue. In Fig. 8a, the PSO-MOA model has the highest budget reduction (25-40 percent) while the QoI-heu model has a budget reduction between 10 and 26 percent of its original budget. PSO-QoI model presents a budget reduction of 5 percent when the number of tasks is less than 50 tasks. For more than 50 tasks, PSO-QoI model presents a budget increase and is an increasing function of the number of tasks. From Fig. 8b, it can be observed that the PSO-MOA model has an increase of the allocation rate between 0 and 15 percent of the original task allocation rate (Fig. 7a). For the case of 150 tasks, the multiple tasks allocation mechanism allows the PSO-MOA model to improve the task allocation rate to 95 percent while PSO-QoI model presents a task allocation rate gain of 30 percent of its original task allocation rate (i.e., 62 percent).

6.2.2 Priority Queue

Fig. 9 presents the impact of the queue size in the task manager side. In particular, Figs. 9a and b present the average waiting time and the successful completion rate for those tasks being queued due to the unavailability of workers to perform them upon their arrival. As we can see, our model has higher rate of successful number of tasks that have been queued and completed than the other two benchmark models. Moreover, the average waiting time is lower than the benchmark models. Both benchmark models present higher waiting time in queue that the PSO-MOA model while their rate of task being successful queued are lower in comparison to our model (see Fig. 9c).

As expected, the priority queue scheme is indeed leveraged by our multi-objective task allocation algorithm. This is owing to the fact that our algorithm also aims at minimizing the response time to perform every task that were already allocated. Therefore, the workers will be available to perform the queued tasks without causing long waiting times and having a high completion task rate.

TABLE 11
Social Graph for Different Number of Workers

No. Workers	Size of social graph	Avg. Known People
200	2,704	13
400	7,506	18
600	17,956	29
800	26,886	33
1,000	34,776	34

TABLE 12
Delegation Mechanism versus Worker's Re-Selection

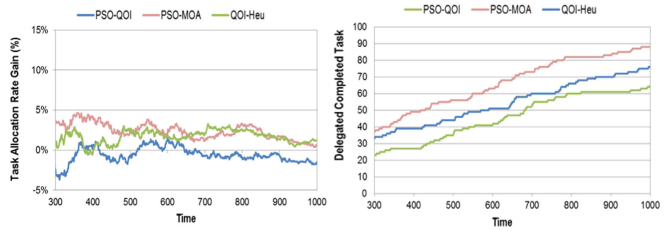
Iteration:	600			1,000		
Model	**Alloc. Rate %**	**Resp. Time (min)**	**Budget $**	**Alloc. Rate %**	**Resp. Time (min)**	**Budget $**
PSO-MOA	80	6	10.2	84	6	10
PSO-MOA-ReSe	81	8.2	11.2	85	9.5	12
PSO-MOA-De	83	7	9.4	85	7.5	9.8

6.3 Delegation Mechanism

The impact of the delegation mechanism over several performance metrics is presented in this section. Although the Foursquare dataset includes social graph information about its users, we could not use it because we extracted a subset of users and its corresponding social network was very limited (approximately two people known per worker) for delegation purposes. Instead, we decided to use the social network from the Enron e-mail dataset.[5] The practice of combining two different datasets is used successfully on other papers [32]. In fact, the two datasets are complementing each other. The Enron email communication network covers all the email communication within a dataset of around half million emails. Nodes of the Enron network are email addresses and if an address i sent at least one email to address j, the graph contains an undirected edge from i to j. The number of nodes is 36,692. Each worker from our subset is mapped to one e-mail address of the Enron e-mail dataset. Table 11 shows the average number of known people per worker after the mapping according to the size of the worker subset. The probability of dropping a task (p) is set to 0.2.

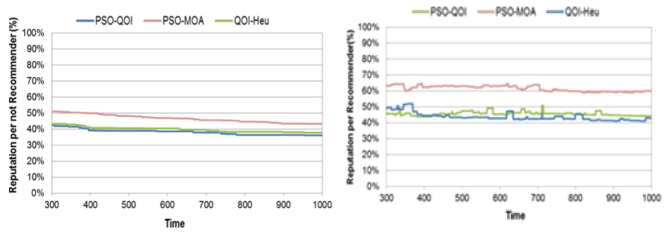
First, we run simulation using PSO-MOA with neither delegation nor reselection (PSO-MOA), with the delegation mechanism (PSO-MOA-DE) and with worker reselection by the MCS system (PSO-MO-ReSe). Table 12 shows the average results for three models at iterations 600 and 1,000. As it can be observed, the task allocation rate for the model that attempts to enhance the task allocation rate of the original model (PSO-MOA) using the delegation scheme presents values similar to the values using the worker re-selection by the MCS system and both schemes enhanced the task allocation rate of the original model. However, the response time and budget are increased for the model using the reselection compared to the one with the delegation mechanism. This is

5. https://snap.stanford.edu/data/email-Enron.html



(a) Gain Task allocation rate (b) Completed delegated tasks

Fig. 10. Impact of the delegation mechanism.



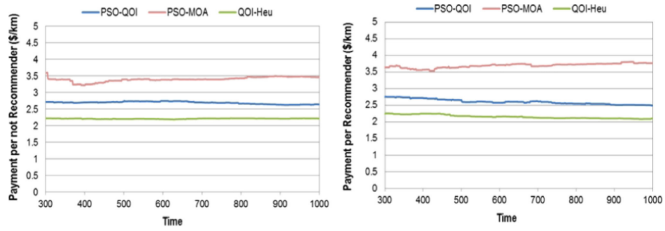
(a) Worker who does not recommend (b) Recommender

Fig. 11. Reputation per worker.

because the MCS system does not select workers among the full set of workers, but instead relies on the worker's recommendation and reduces its search to a subset of workers.

Fig. 10a presents the task allocation rate gain when the delegation mechanism is incorporated to the three models and a worker can be assigned to only one task at a time. It can be noticed the highest task allocation rate gain is usually obtained using the PSO-MOA model. Fig. 10b depicts the number of delegated and completed tasks for each model and shows that the proposed model is able to delegate more tasks than the benchmark models. Fig. 11 presents the average reputation per workers. In particular, these figures show the average reputation for workers who do not recommend and recommenders. It is worth noticing that the proposed delegation mechanism can effectively incentivize a worker to recommend other workers from his social network and allows them to increase their reputation and thanks to this increase they can keep their payment higher than in the other two models (see Fig. 12).

The benchmark models fail to incentivize through the proposed delegation mechanism owing to the fact that these two models require more workers to carry out one task and there is less available workers that can be reached under a recommendation.



(a) Worker who does not recommend (b) Recommender

Fig. 12. Payment per worker.

In summary, the proposed delegation mechanism is able to incentivize workers, guaranteeing their payment by means of keeping their high reputation level. In particular, our model increases the number of tasks being delegated owing to the fact that our model uses less number of workers per task. Therefore, there is a high probability that one worker who does not finish his/her task can affect the total required QOI per task.

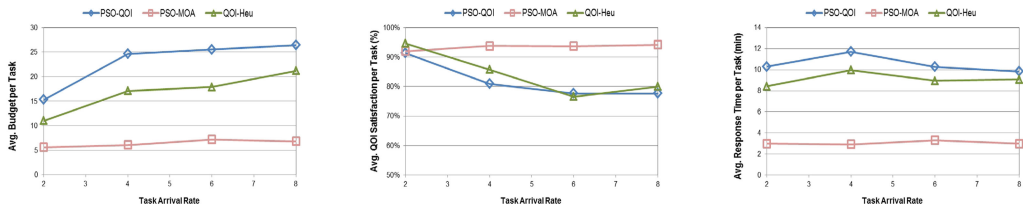
6.4 Evaluation of Each Component of the Realistic Framework

Table 13 presents the impact of each component of the solution on the performance metrics for the three models. The first column indicates the name of the metric. Then, the five consecutive columns present the results for each model. The first column under each model indicates the case where only one task can be carried out per worker without any additional component. The second to fourth columns correspond to the results for each model considering only one component of the proposed framework and the fifth column combines all the components.

Each component leads to an enhancement over the performance metrics in comparison with the model that allocates just one task per worker. In particular, the proposed framework (PSO-MOA) using all components can allocate around 81 percent of the requested tasks with a QoI satisfaction of 90 percent and an estimation error of 19 percent. PSO-QoI model allocates 70 percent of the requested tasks with a QoI satisfaction of 90 percent and an estimation error of 32 percent. The heuristic model allocates 64 percent of the requested tasks with a QoI task satisfaction of 94 percent and an estimation error of 32 percent. As expected, the proposed framework reduces the budget and response time per task around 70 and 50 percent in comparison with the two benchmark models respectively.

TABLE 13
Impact of Each Component over the Performance Metrics

Performance	PSO-QoI					PSO-MOA					QoI-Heu				
Metrics	One Task	Priority Queue	Multipe Tasks	Deleg	All	One Task	Priority Queue	Multipe Tasks	Deleg	All	One Task	Priority Queue	Multipe Tasks	Deleg	All
Estimation Error (%)	28	28	27	31	32	18	19	19	20	19	28	27	29	33	32
Average Reputation (%)	51	51	50	50	51	63	65	62	61	65	49	51	51	49	49
Task Allocation Rate (%)	58	58	69	70	70	79	78	79	80	81	55	59	60	67	64
QoI Satisfaction (%)	86	87	86	90	90	89	89	89	89	90	88	87	88	94	94
Budget ($)	37.40	35.32	33.63	39.04	36.85	11.61	12.72	12.00	11.51	11.88	28.58	27.954	28.55	29.46	30.23
Time (min)	18.79	18.81	16.93	21.36	21.53	5.64	7.87	5.66	5.96	8.32	17.42	18.43	17.00	20.39	22.09
Running Time (min)	1.15	1.18	1.17	1.06	1.11	1.42	1.55	1.50	1.46	1.41	0.05	0.04	0.05	0.04	0.05



(a) Budget (b) QOI Satisfacion (c) Time

Fig. 13. Performance metrics for variable task arrival rate.

Finally, the running times for PSO-based models are higher than the heuristic model (1.1 and 1.5 minutes for the PSO-QoI and PSO-MOA models respectively). In particular, PSO-MOA model presents the total response time including the response time and running time lower than the one for the QoI-Heu model. Moreover, the running time for the PSO-based algorithms can be further reduced using a PSO variant. However, this is out of the scope of the paper and it can be investigated as future work.

6.5 Impact of Variable Task Arrival Rate

Here, the performance of the three models when the MCS system reaches the steady state conditions is presented. Fig. 13 shows the quality of information, the budget and the average response time per task for variable task arrival rate λ (i.e., 2-8 tasks/min). For this scenario, the task manager queue size Q was set up to 5, the worker queue size is equal to 2, the delegation mechanism is used and available number of workers is equal to 1,000. As expected, our framework can effectively reduce the budget and response time. It can be noticed that as the task arrival rate increases, the QoI for the benchmark models decreases, while in our model slightly increases.

In summary, our solution reduces the budget by 5 to 35 percent and accommodates around 20 percent more tasks than the benchmark models. Moreover, its response time is lower than the response time of the QPI-Heu and PSO-QoI models. It is also shown that the MCS framework encourages the workers through the delegation mechanism by keeping the level of workers' reputation high. Finally, the queuing schemes allow the framework to further increase the task allocation rate without compromising the required response time. Therefore, our MCS framework increases the average quality of information per task, reduces the budget and minimizes the response time.

7 Conclusion

A service computing framework for task management in MCS systems is introduced. It consists of a multi-objective task allocation algorithm, queuing schemes and a delegation mechanism. The multi-objective task allocation algorithm was implemented using PSO to find a compromise between the aggregated quality of information/budget ratio and the response time. This algorithm was compared with two algorithms: one meta-heuristic (PSO-QoI) and one heuristic (QoI-Heu) that aim at maximizing the QoI per task under an incremental scenario and a realistic scenario. For the incremental scenario, our solution reduces the budget between 5 and 35 percent, accommodates around 20 percent more tasks than the benchmark models while requiring less time for the data collection. For the realistic scenario, the proposed framework provides incentives to the workers through the delegation mechanism. In fact, our model presents an average recommenders' reputation higher than 58 percent while the other two models present a reputation lower than 50 percent. Furthermore, the queuing schemes allow the proposed framework to further increase the allocation rate without compromising the required response time. As future work, we propose to investigate the optimization of the workers' route when allocated to several tasks. We also propose to investigate the implementation of other delegation mechanisms and the analysis of the framework performance under mobile users arrival/departure processes to determine when the engagement strategies are needed to guarantee the QoI per task.

Acknowledgments

This work has been partially funded by Khalifa University Internal Research Level 1, fund code 210068.

References

[1] B. Guo, et al., "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm," *ACM Comput. Surveys*, vol. 48, no. 1, pp. 7:1–7:31, 2015.

[2] V. Coric and M. Gruteser, "Crowdsensing maps of on-street parking spaces," in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst.*, May 2013, pp. 115–122.

[3] N. Maisonneuve, M. Stevens, M. Niessen, and L. Steels, "Noisetube: Measuring and mapping noise pollution with mobile phones," in *Proc. Inf. Technol. Environmental Eng.*, 2009, pp. 215–228.

[4] B. Guo, H. Chen, Z. Yu, X. Xie, S. Huangfu, and D. Zhang, "FlierMeet: A mobile crowdsensing system for cross-space public information reposting, tagging, and sharing," *IEEE Trans. Mobile Comput.*, vol. 14, no. 10, pp. 2020–2033, Oct. 2015.

[5] X. Zhang, et al., "Incentives for mobile crowd sensing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 54–67, Jan./Mar. 2016.

[6] M. H. Cheung, R. Southwell, F. Hou, and J. Huang, "Distributed time-sensitive task selection in mobile crowdsensing," in *Proc. 16th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2015, pp. 157–166.

[7] Y. Liu, B. Guo, Y. Wang, W. Wu, Z. Yu, and D. Zhang, "TaskMe: Multi-task allocation in mobile crowd sensing," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 403–414.

[8] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han, "ActiveCrowd: A framework for optimized multitask allocation in mobile crowdsensing systems," *IEEE Trans. Human-Mach. Syst.*, vol. 47, no. 3, pp. 392–403, Jun. 2017.

[9] I. Koutsopoulos, "Optimal incentive-driven design of participatory sensing systems," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1402–1410.

[10] N. Do, C.-H. Hsu, and N. Venkatasubramanian, "CrowdMAC: A crowdsourcing system for mobile access," in *Proc. 13th Int. Middleware Conf.*, 2012, pp. 1–20.

[11] H. Shah-Mansouri and V. Wong, "Profit maximization in mobile crowdsourcing: A truthful auction mechanism," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 3216–3221.

[12] H. Yu, C. Miao, Z. Shen, C. Leung, Y. Chen, and Q. Yang, "Efficient task sub-delegation for crowdsourcing," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 1305–1312.
[13] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor, "AutoMan: A platform for integrating human-based and digital computation," *Commun. ACM*, vol. 59, no. 6, pp. 102–109, 2016.
[14] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation*, 4th ed. Englewood Cliffs, NJ, USA: Prentice Hall, Dec. 2004.
[15] C. Zhou, C.-K. Tham, and M. Motani, "QOATA: Qoi-aware task allocation scheme for mobile crowdsensing under limited budget," in *Proc. IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, Apr. 2015, pp. 1–6.
[16] H. Yu, C. Miao, Z. Shen, and C. Leung, "Quality and budget aware task allocation for spatial crowdsourcing," in *Proc. Int. Conf. Autonomous Agents Multiagent Syst.*, 2015, pp. 1689–1690.
[17] J. Liu, H. Shen, and X. Zhang, "A survey of mobile crowdsensing techniques: A critical component for the internet of things," in *Proc. 25th Int. Conf. Comput. Commun. Netw.*, Aug. 2016, pp. 1–6.
[18] L. Jaimes, I. Vergara-Laurens, and M. Labrador, "A location-based incentive mechanism for participatory sensing systems with budget constraints," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, Mar. 2012, pp. 103–108.
[19] L. Mo, et al., "Optimizing plurality for human intelligence tasks," in *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 1929–1938.
[20] L. Varshney, J. Rhim, K. Varshney, and V. Goyal, "Categorical decision making by people, committees, and crowds," in *Proc. Inf. Theory Appl. Workshop*, Feb. 2011, pp. 1–10.
[21] J. Surowiecki, *The Wisdom of Crowds*. New York, NY, USA: Anchor, 2005.
[22] Foursquare dataset, 2013. [Online]. Available: https://archive.org/details/201309_foursquare_dataset_umn
[23] V. D. Blondel, et al., "Data for development: The D4D challenge on mobile phone data," *CoRR*, 2012. [Online]. Available: http://arxiv.org/abs/1210.0137
[24] G. A. Korn and T. M. Korn, *Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review*. Mineola, NY, USA: Dover Publications, 2000.
[25] S. He, D.-H. Shin, J. Zhang, and J. Chen, "Toward optimal allocation of location dependent tasks in crowdsensing," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 745–753.
[26] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[27] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Adv. Eng. Inform.*, vol. 19, no. 1, pp. 43–53, 2005.
[28] Perez. R. and K. Behdinan, "Particle swarm optimization in structural design," in *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization*. Vienna, Austria: Itech Education and Publishing, 2007, pp. 532–555.
[29] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Proc. IEEE Swarm Intell. Symp.*, 2007, pp. 120–127.
[30] F. Lobo, *Parameter Setting in Evolutionary Algorithms*. Berlin, Germany: Springer, 2007.
[31] P. Tannenbaum, *Excursions in Modern Mathematics*. Cambridge, U.K.: Pearson, 2013.
[32] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Optimal load distribution for the detection of VM-based DDoS attacks in the cloud," *IEEE Trans. Serv. Comput.*, 2017.

Rebeca Estrada received the BSc degree in computer engineering from ESPOL, in 1995, the master's degree with specialization in telecommunication from ITESM, Monterrey, México, in 1998, and the engineering doctoral degree from École de Technologie Supérieure, University of Quebec, in 2014. She holds an associate professor position in the department of Electrical Engineering and Computer Science at Escuela Superior Politécnica del Litoral (ESPOL), Guayaquil, Ecuador. Currently, she is leading a research group (ReDIT) with focus on Networking and Technological infrastructure at ESPOL. Her research work is oriented to resource allocation in mobile crowd-sensing systems, cloud computing systems and two-tier wireless network.

Rabeb Mizouni received the MSc and PhD degrees in electrical and computer engineering from Concordia University, Montreal, Canada, in 2002 and 2007, respectively. She is an assistant professor in electrical and computer engineering with Khalifa University. Currently, she is interested in the deployment of context aware mobile applications, crowd sensing, software product line and cloud computing.

Hadi Otrok received the PhD degree in ECE from Concordia University. He holds an associate professor position in the Department of ECE, Khalifa University, an affiliate associate professor in the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, and an affiliate associate professor in the electrical department at cole de Technologie Suprieure (ETS), Montreal, Canada. He is an associate editor at: *Ad-Hoc Networks* (Elsevier), the *IEEE Communications Letters*, *Wireless Communications and Mobile Computing* (Wiley). He co-chaired several committees at various IEEE conferences. He is an expert in the domain of computer and network security, web services, ad hoc networks, application of game theory, and cloud security. He is a senior member of the IEEE.

Anis Ouali received the BSc degree in computer engineering from LEcole Nationale des Sciences de l'Informatique (ENSI), Tunisia, in 2000, the MSc degree in computer science from Universite du Quebec A Montreal (UQAM), Canada, in 2004, and the PhD degree from the Electrical and Computer Engineering Department, Concordia University, Montreal, Canada, in 2011. His research interests include P2P networks for video streaming, distributed computing and content adaptation. He joined EBTIC in 2010 and is currently working in the network optimization team which focuses on solving network design related problem.

Jamal Bentahar received the PhD degree in computer science and software engineering from Laval University, Canada, in 2005. He is a full professor with Concordia Institute for Information Systems Engineering, Faculty of Engineering and Computer Science, Concordia University, Canada. From 2005 to 2006, he was a postdoctoral fellow with Laval University, and then Simon Fraser University, Canada. His research interests include services computing, applied game theory, computational logics, model checking, multi-agent systems, and software engineering. He is a member of the IEEE.

A Reinforcement Learning Method for Constraint-Satisfied Services Composition

Lifang Ren, Wenjian Wang, and Hang Xu

Abstract—With increasing adoption and presence of Web services, service composition becomes an effective way to construct software applications. Composite services need to satisfy both the functional and the non-functional requirements. Traditional methods usually assume that the quality of service (QoS) and the behaviors of services are deterministic, and they execute the composite service after all the component services are selected. It is difficult to guarantee the satisfaction of user constraints and the successful execution of the composite service. This paper models the constraint-satisfied service composition (CSSC) problem as a Markov decision process (MDP), namely CSSC-MDP, and designs a Q-learning algorithm to solve the model. CSSC-MDP takes the uncertainty of QoS and service behavior into account, and selects a component service after the execution of previous services. Thus, CSSC-MDP can select the globally optimal service based on the constraints which need the following services to satisfy. In the case of selected service failure, CSSC-MDP can timely provide the optimal alternative service. Simulation experiments show that the proposed method can successfully solve the CSSC problem of different sizes. Comparing with three representative methods, CSSC-MDP has obvious advantages, especially in terms of the success rate of service composition.

Index Terms—Web service composition, constraint-satisfied, uncertainty of service behaviors, undetermined QoS, Markov decision process (MDP), Q-learning algorithm

1 Introduction

With the rapid development of cloud computing, services are emerging as a powerful vehicle for organizations to deliver their applications over the Internet. As the number of services increases, software applications are no longer built from scratch, but rather through integration of available services distributed in the Web, which leads to the service composition. In this way, component services can be integrated into more capable composite services to fulfill more and more complex demands of users. Therefore, it is an inevitable trend to integrate the available services to meet various requirements from users [1], [2].

It is expected that there will be an increasing number of services with the same functionality and different quality of service (QoS), such as response time, availability, reliability, throughput, price, success rate, and so on. Nevertheless, the Internet environment is dynamic and the service evolutions take place erratically, which leads to the uncertain QoS and service behavior. Such as, the increase in network traffic will bring a prolonged response time, and sometimes a service is temporarily unavailable due to the upgrade evolution. All these make service composition a complex task. Along with the fact that users tend to propose different constraints on the QoS of the composite service. For example, users may expect the response time of a composite service to be less than a certain threshold while the execution cost of the composite service must fall within a budget. However, in general there is a tradeoff between cost and response time (or other QoS attributes) for a composite service. The constraints from user further increase the difficulties of service composition. Given the above consideration, it is important and challenging to design a constraint-satisfied service composition (CSSC) method adaptive to the uncertainty QoS and service behavior [3].

So far, many research efforts have been devoted to solving the CSSC problem [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. All these methods regard the QoS values as being determined, and they divide the process of service composition into component services selection stage and composite service execution stage, which means that the composite service are executed after all the component services are selected. However, as the supporting environment, the Internet is highly dynamic, and service evolutions take place casually, thus the QoS values of the same service invoked by the same user at different times can be very different. Due to the uncertainty of QoS and service behavior, it is difficult to guarantee that the optimal component service in the selection stage is still optimal during the execution stage. Even worse, the selected optimal component service may become unavailable during the execution of composite service. Thus, the optimization process has to be performed again, however, the

- *W. Wang and H. Xu are with the School of Computer and Information Technology, Shanxi University, Taiyuan 030006, PR China. E-mail: wjwang@sxu.edu.cn, xuh102@126.com.*
- *L. Ren is with the School of Computer and Information Technology, Shanxi University, Taiyuan 030006, PR China and with the School of Applied Mathematics, Shanxi University of Finance and Economics, Taiyuan 030006, PR China. E-mail: renlf@sxufe.edu.cn.*

Manuscript received 4 July 2016; revised 4 June 2017; accepted 9 July 2017. Date of publication 13 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Wenjian Wang.) Digital Object Identifier no. 10.1109/TSC.2017.2727050

re-optimization cannot guarantee the successful execution of the new optimal composite service.

The goal of CSSC is to find and execute the most appropriate service for each task in the business workflow, so far as to satisfy the functional requirements and QoS constraints as possible as it can be. However, service behavior is uncertain and QoS values are variable, this results to difficulties in attempting to satisfy user constraints optimally and robustly. Consequently, the CSSC process can be considered as an optimization problem of multi-stage decisions within an uncertain decision-making environment. The Markov decision process (MDP), as a model of the reinforcement learning, is a theoretical tool for studying the optimization problem of the multi-stage decision process in stochastic environment [15], so it is especially suitable for solving the CSSC problem. Accordingly, this paper models the CSSC problem as an MDP, namely CSSC-MDP, and designs a Q-learning algorithm to solve the model. By this way, CSSC-MDP integrates the advantages of global optimization approaches and local optimization approaches. The main benefits of CSSC-MDP are as follows:

- CSSC-MDP selects component services during the composite service execution, thus it avoids the failure of whole composite service caused by the failure of a component service. Therefore, CSSC-MDP is robust.
- CSSC-MDP selects the optimal candidate service for a task based on the constraints which need to be satisfied by the following services. Hence, CSSC-MDP is self-adaptive to the uncertain QoS values.
- The selection strategy of CSSC-MDP aims at maximizing the expected cumulative reward which is on behalf of the satisfaction degree of the user constraints. So, CSSC-MDP is globally optimized.

The remainder of this paper is organized as follows. Section 2 gives the details of our CSSC-MDP approach. In Section 3, an illustrative example is presented to explain concretely the process of CSSC-MDP method. Section 4 reports and analyzes our experimental results. Section 5 overviews the related work. Finally, conclusions are given in Section 6.

2 The CSSC-MDP

In this section, the CSSC problem is formally described at first; second, the approach of this paper is presented; third, some relevant definitions are formalized; next, the CSSC is modeled as an MDP; then, the decision criterion of CSSC-MDP is presented; finally, the Q-learning algorithm to solve the CSSC-MDP is proposed.

2.1 Problem Description

The aim of CSSC is to find the appropriate component services which can be integrated as an optimal composite service, so as to best meet the user's functional and nonfunctional requirements. In this paper, the functional requirements are described as a workflow. Generally, the workflow can be sequential, conditional, parallel and iterative constructs, or more generically, the combination of these structures. However, in fact, service compositions

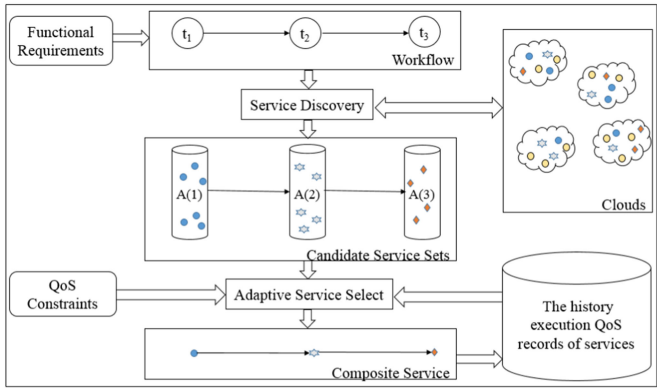


Fig. 1. The schematic of CSSC.

which involve loops, branches or parallel structures can be converted into sequence structures [16]. Moreover, many studies have been done to compute the QoS of composite service in different structures [12], [13]. Therefore, this paper focuses on the CSSC problem with sequential workflow model. Hence, the process of service composition is to select the component services most suitable for each task in the workflow to form the optimal composite service, meanwhile, all the QoS constraints should be satisfied.

Fig. 1 is the schematic of CSSC. At first, the request for service composition from the user is submitted, it includes the functional requirements and QoS constraints. According to functional requirements, a workflow is built, which is the abstract representation of the composite service. For example, in Fig. 1 the workflow is formed with three sequentially executed abstract tasks (denoted as t_1, t_2 and t_3). In the Internet, there exist many service providers, each of them provides some different services. In Fig. 1, a cloud represents a service provider, different shapes within it represent different services it can provide; and the same shape indicates the same functionality. After service discovery, services with the same functionality are gathered into a candidate service set, such as $A(1)$, $A(2)$, and $A(3)$ in Fig. 1. Thus, each $A(i)$ is the alternative services collection for the abstract task t_i. Then, according to the QoS constraints and the historical execution QoS records of candidate services, one service is selected from each $A(i)$ to form the composite service which can satisfy both the functional and the QoS requirements.

Our work in this paper is to find the appropriate service for each task in the workflow (which meets the user's functional requirements) to form an optimal composite service, which satisfies the user's QoS constraints as well as possible. The main difficulties lie in:

- There are contradictions among QoS attributes. Such as a service with shorter response time may have a higher price, and a lower price may as a result of a lower reliability.
- Because the Internet is dynamic, the QoS values of service are undetermined, which leads to the uncertainty of QoS constraints satisfaction.
- There exists the possibility that a component service fails during the execution of composite service. The failure of component service results in the failure of service composition.

This paper studies how these challenges are dealt with.

2.2 Approach Overview

The approach of CSSC-MDP can be described as Algorithm 1.

Algorithm 1. The CSSC-MDP

Input:
The function requirements from the user;
The QoS constraints from the user;
Output:
The execution of composite service and QoS records;
Or the negotiation failure information;
1: Build the workflow template;
2: Discover the available services for each abstract task and gather the historical execution QoS records for them.
3: Evaluate the rationality of user constraints.
4: **if** the constraints are not rational **then**
5: Negotiate with the user.
6: **if** No mutually approved constraints **then**
7: Output: Inappropriate QoS constraints, and Exit.
8: **end if**
9: **end if**
10: Build the service composition CSSC-MDP model;
11: Use the Q-learning algorithm to solve the CSSC-MDP model;
12: Send the observed QoS values to the historical records.

At first, according to user's functional requirements, the workflow template that represents the abstract service composition are built. Then, the available services for each abstract task in the workflow are discovered and collected. Next, the historical execution QoS records for all candidate services are gathered. There are many works studying the construction of workflow [17], [18] and service discovery [19], [20], so this paper does not discuss the implement of these two steps. And after that, the rationality of user constraints will be evaluated; the evaluation in this paper is based on the 3σ principle; for details, please see Section 3.1. If the constraints are rational, continue to the next step; otherwise, negotiate with the user. If the mutually approved constraints are obtained, proceed to the next step; otherwise, abort. After the mutually approved constraints have been obtained, build the service composition CSSC-MDP model; for details see Section 2.4. Then, the Q-learning algorithm are used to solve the CSSC-MDP; for details, see Section 2.6. Based on the learning result, i.e., matrix Q, the component services are selected and executed until all the tasks in the workflow have been completed; Finally, the observed QoS values are sent to the historical records.

In so doing, some benefits include:

- CSSC-MDP avoids the blind service composition where the rationality of user constraints is not taken into account;
- CSSC-MDP can self-adaptive to the variable QoS and undetermined service behaviors;
- It is effective and efficient to solve CSSC-MDP with the Q-learning.

2.3 Definitions Formalization

In this section, we will give the formal description of some definitions relevant to CSSC-MDP.

Definition 1 (Service). *A service is a functionally complete and self-governed resource that can be published, located, and visited through the Web. A service can be formalized as a 4-tuple* $(ID, FunC, QoSE, QoSR)$*, where:*

ID *is the identifier of a service. A service can be uniquely determined by its* ID*.*

FunC *represents the functional class of a service, and it is a triple* (F, I, O)*, where* F *is the functional description of the service,* I *represents the input items of service and* O *represents the output items of the service.*

QoSE *is the expected QoS values offered by the service provider in service description, and it can be formalized as a vector* $(v^{(1)}, v^{(2)}, \ldots, v^{(d)})$*, where* d *is the number of* QoS *attributes which we are concerned with.*

QoSR *is a container of the historical execution* QoS *records of the service. It is formalized as a* $d \times l$ *matrix, where* l *is the historical execution number of the service. The* k*th row of matrix* QoSR *is a vector* $(v_1^{(k)}, v_2^{(k)}, \ldots, v_l^{(k)})$ *that contains* l *historical recorded values of the* k*th* QoS *attribute. The* h*th column of* QoSR *is a vector* $(v_h^{(1)}, v_h^{(2)}, \ldots, v_h^{(d)})$ *that contains all the* d QoS *attributes' values of the* h*th execution of the service.*

For simplicity, in this paper, we give special focus to two commonly used QoS attributes, the response time and the price, but our approach is equally valid for other QoS attributes.

Definition 2 (Candidate service set). *The candidate service set is a collection of alternative services which provide the same functionality but different QoS. In other words, services in such a set have the same* $FunC$ *but differ in QoS. A candidate service set can be formalized as a set* $\{ws_1, ws_2, \ldots, ws_m\}$*, where* m *is the number of services with the desired* $FunC$*, and* $ws_i s (i = 1, 2, \ldots, m)$ *are the IDs of candidate services.*

Definition 3 (Workflow). *Workflow is an abstract description of the business rules. A sequential execution workflow can be formalized as a sequence* $(t_1, t_2, \ldots, t_n)$*, where* $t_i (i = 1, 2, \ldots, n)$ *is the* i*th abstract task, and* n *is the total number of tasks.*

The power of the services lies in that it can be dynamically integrated to execute a new and more complex task. This process is known as *service composition*.

Definition 4 (Composite service). *In order to meet some complex functional requirements, according to certain business logic, services with different functions are integrated into a scalable, loosely coupled, value-added application, namely* composite service. *A composite service can be formalized as a sequence* $(ws_1, ws_2, \ldots, ws_n)$*, where* $ws_i s$ $(i = 1, 2, \ldots, n)$ *are sequently the IDs of services which compose the composite service and* n *is the number of services that compose the composite service.*

Definition 5 (Constraint). *The constraint is usually a QoS requirement about the composite service from the user, such as maximum total price, minimum overall throughput, average response time, etc. A constraint can be expressed in terms of the upper or lower bound of the composite service QoS attribute. Hence, a constraint can be expressed as a triple* $(att \quad opr \quad bnd)$*, where* att *is one of the QoS attributes of the user's concern;* opr *represents relational operators such as* $>, <, \geq, \leq$*, etc.; and* bnd *is the bound. For example,* $(price < 1000)$ *represents the user requires the total cost of the composite service is less than 1,000 monetary unit. In general, user may impose more than one constraint to the composite service, thus the constraints form a constraint set.*

2.4 CSSC-MDP Model

Suppose the QoS constraints from user are considered to be rational; then, based on the MDP model, the constraint-satisfied service composition model CSSC-MDP can be formalized as a 5-tuple (t, S, A, T, R), where:

t is the stage of decision-making; that is, it is the numeric sequence of the task which is being executed. Hence, its possible values are from 1 to the total number of tasks n.

S is the execution state set of the composition service. Consulting literature [21] about the service classifications, we set the four-level-state denoted as $S = \{1, 2, 3, 4\}$. Level 1 represents the user's requirements are excellently satisfied; level 2 means that the composite service can satisfy the user's requirements well; level 3 means that the composite service can basically meet the user's requirements; level 4 indicates the failure of service execution or the violation of user constraints. The number of levels may be determined according to the actual requirements, the details of state division in this paper is given in Section 3.2.

For each concerning attribute, we need to determine the state of composite service. In order to obtain a better performance, we use the worst state of all the QoS attributes as the integrated state, i.e., $s_i = \max\{s_i^{(1)}, s_i^{(2)}, , s_i^{(d)}\}$, where s_i is the state of the service composition after task t_i is finished.

$A = \bigcup_{i=1}^{n} A(t_i)$ is the set of all the candidate services in the model, where $A(t_i) = \{ws_{i1}, ws_{i2}, \ldots, ws_{ik_i}\}$ is the candidate service set for task t_i, and k_i is the total number of services in t_i's candidate set. A is the union of all the $A(t_i)$s; in other words, A contains all the services that may be used in the composite service.

$T(s, ws, s') = Pr(s_{t+1} = s' | s_t = s, a_{t_i} = ws)$ is the probability that the execution of service ws at state s for task t will lead to state s'. This can be calculated according to the historical execution QoS records for service ws.

$R(s', s, ws)$ is the reward function. it is a real-valued function, and $r = R(s', s, ws)$ is the immediate reward received from the state transition from s to s' after ws is executed. We can say that it is used to quantify the benefit of executing service ws, which leads to the state transition. When $r > 0$, it indicates rewards; when $r < 0$, it indicates penalties. After the execution of ws, the greater the difference of $s - s'$, the higher the value of r, and vice versa. The goal of service composition is to select the optimal services to compose the composite service with the highest cumulative reward. The reward function can be defined according to the actual situation, the definition of $R(s', s, ws)$ in this paper can be found in Section 3.3.

In contrast with existing methods, CSSC-MDP considers the state of constraints been satisfied after each service execution. In case of the violation of QoS constraints or the service failure, the stage t remains its value and CSSC-MDP will redo the selection and execution. Otherwise, let $t = t + 1$, and select the service which can lead to an optimal composite service and execute it for the next task. The process of decision-making is described in detail in the following section.

2.5 Decision-Making of CSSC-MDP

The core problem of an MDP is to determine the optimal policy π^* that will maximize the cumulative function of the rewards. The function $\pi(s)$ specifies the action that the agent will choose in state s. For the CSSC-MDP model, the optimal service of a task is selected based on the constraints state and the historical execution QoS records of candidate services.

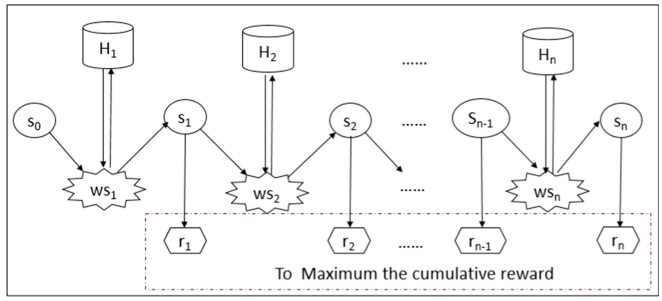


Fig. 2. The decision-making process in CSSC-MDP.

Fig. 2 shows the decision-making process in CSSC-MDP. At the initial state s_0, according to the historical execution QoS recorded in H_1 for services in $A(1)$, for task t_1 the agent selects and executes the optimal service denoted as ws_1, the constraints state transfers to a new state, denoted as s_1, and the execution QoS data are then appended to the historical records for ws_1 in H_1. Based on the new constraints state s_1, the immediate rewards r_1 can be calculated. According to the historical execution QoS recorded in H_2 for services in $A(2)$, for task t_2 the agent selects and executes the optimal service at state s_2, and so forth, until all the tasks in the business workflow have been done.

The strategy of CSSC-MDP is a mapping from state space to service space. To determine the optimal strategy that leads to the optimal composite service, we need to define the value function $V_t(s)$ to calculate the expected cumulative rewards of the service execution for task t at state s. The value function $V_t(s)$ is defined by the recursion formula $V_t(s) = \sum_{s'} T(s, ws, s')\big(R(s, s', ws) + \gamma V_{t+1}(s')\big)$, where s' is one of the possible constraint states after the execution of the service ws; $T(s, ws, s')$ is the transition probability from state s to s' caused by service ws; $R(s, s', ws)$ is the immediate reward of the execution of service ws; and γ is the discount factor for future rewards. Consequently, $V_t(s)$ is the cumulative reward resulting from the execution of service ws at task t in state s.

The strategy that leads to the highest value of $V_t(s)$ is the optimal strategy, which can be expressed as $\Pi_t(s) = \arg\max_{ws \in A(t)} V_t(s)$.

Under the guidance of the optimal strategy, the decision-making process of CSSC-MDP maximizes the exceptional cumulative rewards.

CSSC-MDP selects a service for a task after the previous task is completed, hence it can update the state according to the observed QoS values of services which have been completed. That is to say, CSSC-MDP can more realistically reflect the QoS constraints which need the following service to satisfy, and can provide a reliable guarantee for the next service selection to meet the user's constraints. This process continues until all the tasks have been accomplished in turn.

In theory, after modeling the CSSC problem as an MDP, the model can select the optimal strategy dynamically, so the service composition satisfies the user with real-time globally optimal results. However, the computational time complexity of the exact solution of an MDP problem has

been proved to be completely P [22]. Hence, what follows is a Q-learning algorithm to solve the CSSC-MDP model.

2.6 CSSC-MDP Solving

In practice, the existing exact methods to solve the MDP can only accurately solve small problems [15]. Q-learning is a model-free reinforcement learning method that uses rewards to reinforce actions which lead the model to a better state; hence, it can help to learn the optimal policy in a stochastic environment [16]. Therefore, we use Q-learning to solve the CSSC-MDP model. In Q-learning, the transition probabilities are generally obtained through many simulations. However, because services are mostly commercial, it is unlikely to invoke services to test their performance. Hence, in this paper we learn the transition probabilities of service from their historical execution records. Algorithm 2 is the Q-learning algorithm for CSSC-MDP.

Algorithm 2. Q-Learning Algorithm for CSSC-MDP

Input:
The total number of sequential tasks n;
The maximal size of the candidate service sets m;
The user constraints C_0;
The historical execution QoS records for all services H;
Output:
The result of Q-learning, i.e., matrix Q;

```
Initial Q ← zeros(3 × n, m);
while Q is not convergent do
  Initial variables: t ← 1, s_0 ← 1, C ← C_0;
  while t ≤ n do
    Use the ε-greed policy to choose ws ∈ A(t) based on s_0;
    if ws.QoSR ≠ Φ then
      Randomly choose an execution historical QoS record
      h of service ws;
    else
      h = ws.QoSE
    end if
    Based on h and the current constraints C, calculate the
    state s;
    Based on s and s_0, calculate the immediate rewards r;
    if s < 4 then
      Q(3(t−1)+s_0, ws) ← (1−α)Q(3(t−1)+s_0, ws) + α[r +
      γ max_{ws'∈A(t+1)} Q(3t+s_0, ws')];
      According to h, update C;
      t ← t+1; s_0 ← s;
    else
      Q(3(t−1)+s_0, ws) ← (1−α)Q(3(t−1)+s_0, ws) + αr;
      t ← 1; s_0 ← 1; C ← C_0;
    end if
  end while
end while
return Q;
```

As shown in Algorithm 2, we first initialize Q as a matrix with $3 \times n$ rows and m columns whose elements are all zeros, where n is the total number of sequential tasks, and m is the maximal size of the candidate service sets. For every service, we need to consider its performance at different states; hence, each row of matrix Q represents the scores of a candidate service at one state. For example, the 5th row of matrix Q represents the scores of the candidate services of task t_2 at state 2. Because state 4 represents service failure or constraint violation, that is, the service composition cannot proceed from state level 4, in the matrix Q, we only consider the performance scores of services in the first 3 states.

Next, an iteration process is performed to learn the matrix Q. At the first iteration, we initialize the task sequence number t to 1, set the initial state s_0 to 1 (we assume that service compositions always start from the best state), and use the original constraints C_0 to initialize the current constraints C. Then for each task from the first to the last, we use an ε-greed strategy to select one service ws from its candidate service set $A(t)$, i.e., when the random number is smaller than ε, the greed strategy is adopted, otherwise, randomly selects a service. On the one hand the ε-greed can prevent the algorithm from premature convergence, on the other hand it gives chances to the newcomer in service selection. For the selected service ws, if it has been invoked, we randomly choose one piece of the QoS record h from the execution historical records of service ws and consider h as the currently executing QoS. We don't choose the best one, the worst one, the average one or the latest one, but choose the random one piece of QoS record is to simulate the various possible behaviors of service. This is because we want to learn the most likely performance of service from all random behaviors of services. Otherwise if ws has no QoS record, we take the expected QoS $ws.QoSE$ given by service provider as its QoS record h. Based on the QoS information of historical record h and the current constraints C, we calculate the constraint-satisfied state s as described in detail in the Section 3.2. According to the new state s and the old state s_0, we calculate the immediate rewards r. Details of the method can also be found in the Section 2.4. If the selected service ws executes successfully and the user constraints are not violated, we update the element of Q at the tth row and the wsth column, where we use α $(0 \leq \alpha \leq 1)$ as the learning rate; that is, the new $Q(t, ws)$ is composed of $(1-\alpha)$ original $Q(t, ws)$ and α new rewards. The new rewards include the immediate reward and the expected rewards in the future. The γ in the formula is the discount factor, means that the rewards in the future are not necessarily as important as the immediate rewards, and $\gamma \in [0, 1]$ can be inferred. Then, we update the current constraints based on the QoS values of the selected QoS record h. For example, the response time constraint will be reduced by the response time of h. After all these steps, we update the task sequence number t and use the current state s to update the original state s_0. In the circumstances, either service failure or constraint violation, i.e., at the state $s = 4$, the immediate reward turns into a penalty. And because the composition is interrupted, the expected rewards in the future are not included in the $Q(t, ws)$. Also for the same reason, t, s_0, and C are returned to their initial values. The iteration does not terminate until matrix Q is convergent.

Q-learning algorithm learns the ability to satisfy the constraints for every service at every state. Every element of matrix Q is the score of this ability. Based on the matrix Q and the user constraints, service composition can be implemented. The main benefits of Q-learning include:

- The Q-learning algorithm can learn the ability of services to satisfy the constraints from the historical QoS records, without redundant service execution;

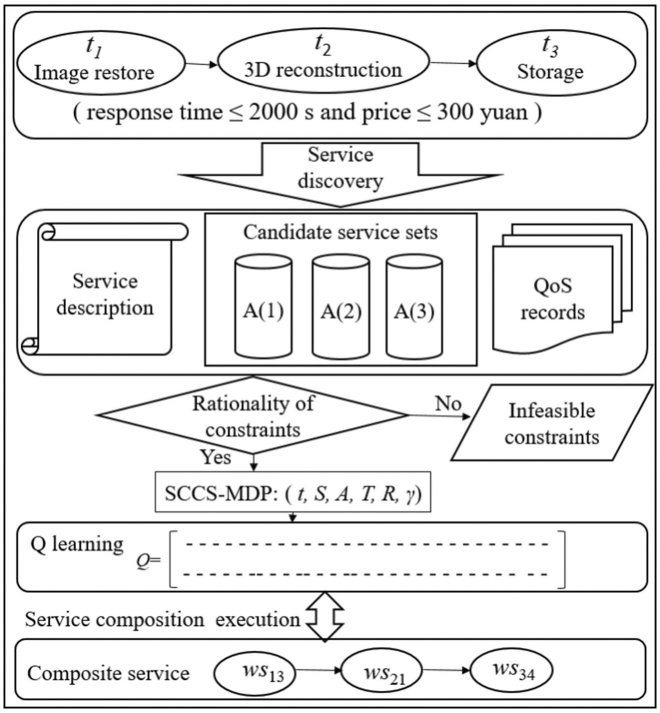


Fig. 3. The process of CSSC-MDP.

- It doesn't need to define transaction function, which can be implied by the Q-learning process automatically;
- The ε-greedy selection strategy in the Q-learning not only makes the algorithm more efficient but also guarantees every service has the opportunity to be executed.

3 An Illustrative Example

This section illustrates the approach of CSSC-MDP through an example. Consider the following CSSC scenario: a user captured a set of multi-angle photos of an object in motion, and he wants to use this set of images to reconstruct the 3D model of the object. Due to limited local resources, the user wants to complete the task by means of Web services provided over the Internet. Moreover, he also expects that the total response time is less than 2,000 seconds and the total price is less than 300 yuan. To meet the user's requirements, we composite the available Web services. Since the images for this moving object are blurred, we first need the image restoration service to restore the blurred 2D images. Then we utilize the computing power provided by Web services to run the 3D reconstruction algorithm to obtain the 3D model of the object. Finally, we store the 3D model in the cloud space for the user. Meanwhile, we should ensure that the user's QoS constraints are best satisfied. Fig. 3 shows the process of service composition in this scenario.

Obviously, the user's functional requirements can be decomposed into three sequential tasks: image restoration task t_1, the 3D reconstruct task t_2 and the cloud storage task t_3. And the composite service should satisfy the following constraints:

$$(ResponseTime \leq 2000) \text{ and } (Price \leq 300). \quad (1)$$

It is a CSSC problem. We need to discover services for each task at first. In the service computing environment, a service provider can provide services with different functions and services with the same function can be provided by different providers. Suppose, after service discovery, as shown in Fig. 3, there are 4 candidate services for the first task t_1 denoted as $A(1) = \{ws_{11}, ws_{12}, ws_{13}, ws_{14}\}$, 3 candidate services for the second task t_2 denoted as $A(2) = \{ws_{21}, ws_{22}, ws_{23}\}$, and 4 candidate services for the third task t_3 denoted as $A(3) = \{ws_{31}, ws_{32}, ws_{33}, ws_{34}\}$. And let's suppose the prices of each service in $A(1)$ are $(82, 90, 70, 85)$, in $A(2)$ are $(140, 230, 180)$, and in $A(3)$ are $(20, 30, 25, 22)$. The historical response time records for each service in $A(1)$ are stored in the following matrix. Values in the kth row of the matrix represent the response time of service ws_{1k} for different execution histories, where -1 means service failure and NaN represents the absence of record.

$$\begin{pmatrix} 417 & -1 & 429 & 416 & 421 & -1 & 414 & 430 & -1 & 421 \\ 320 & 289 & 307 & 322 & 286 & 295 & 313 & 317 & 321 & 296 \\ 390 & 362 & 371 & 380 & 310 & 373 & 367 & -1 & NaN & NaN \\ 434 & 405 & 421 & -1 & 400 & 403 & 413 & NaN & NaN & NaN \end{pmatrix}$$

The historical response time records for each service in $A(2)$ are stored in the following matrix.

$$\begin{pmatrix} 1417 & 1249 & -1 & 1641 & 1521 & -1 & NaN & NaN & NaN & NaN \\ 1201 & 1277 & 1307 & 1021 & 1086 & 1395 & 1313 & 1417 & 1121 & 996 \\ 1450 & 1242 & 1431 & -1 & 1240 & 1433 & 1627 & -1 & NaN & NaN \end{pmatrix}$$

The historical response time records for each service in $A(3)$ are stored in the following matrix.

$$\begin{pmatrix} 173 & 133 & 121 & 146 & -1 & -1 & 119 & 128 & -1 & NaN \\ 90 & 109 & 97 & 112 & 86 & 95 & 113 & 107 & 111 & 106 \\ 112 & 132 & 101 & 130 & -1 & 122 & 109 & 113 & NaN & NaN \\ 174 & 153 & 124 & -1 & 140 & 133 & 113 & NaN & NaN & NaN \end{pmatrix}$$

After service discovery, our approach can be implemented.

3.1 Constraint Rationality Evaluation

First of all, we will evaluate the rationality of user constraints. Based on the historical records, the means and standard variances of both price and response time for each task can be calculated. In the above example, $\mu^{(c)} = (81.75, 216.67, 24.25)$ and $\sigma^{(c)} = (8.50, 70.95, 4.35)$ are means and standard variances of price for each task. $\mu^{(r)} = (368.10, 1319.30, 120.76)$ and $\sigma^{(r)} = (51.30, 181.60, 21.61)$ are means and standard variances of response time for each task. Hence, the mean of price for the composite service is $\mu_{cs}^{(c)} = \sum_{i=1}^{3} \mu_i^{(c)} = 322.67$, and the standard variances of price for the composite service is $\sigma_{cs}^{(c)} = \sqrt{\sum_{i=1}^{3} (\sigma_i^{(c)})^2} = 71.59$, the mean of response time for the composite service is $\mu_{cs}^{(r)} = \sum_{i=1}^{3} \mu_i^{(r)} = 1808.16$, and the standard variances of response time for the composite service is $\sigma_{cs}^{(r)} = \sqrt{\sum_{i=1}^{3} (\sigma_i^{(r)})^2} = 189.94$. Since $\mu_{cs}^{(c)} - 2\sigma_{cs}^{(c)} < 300 < \mu_{cs}^{(c)} + 2\sigma_{cs}^{(c)}$ and $\mu_{cs}^{(r)} - 2\sigma_{cs}^{(r)} < 2000 < \mu_{cs}^{(r)} + 2\sigma_{cs}^{(r)}$, according to the 3σ principle of normal distribution, the constraints can be satisfied with a probability of 0.9544. Hence, we have reason to believe that the user constraints are rational.

TABLE 1
An Execution of the Service Composition

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	380	70	2
2	ws_{21}	1,249	140	2
3	ws_{34}	174	22	1

3.2 State Level Division

To set the levels for services, at first, we determine the median QoS attribution value of every service based on its execution historical QoS records (not include the failure records). Such as, for task t_1, the response time medians of candidate service are $v_1^{(r)} = (421, 307, 371, 408)$, and the mean response time of task t_i can be calculated by $\mu_i^{(r)} = \frac{1}{m}\sum_{j=1}^{m} v_i^{(r)}(j)$. Where m is the total number of candidate services for task i. In this way, we can calculate the mean response time for tasks t_1, t_2 and t_3, $\mu_1^{(r)} = 376.75$, $\mu_2^{(r)} = 1380$ and $\mu_3^{(r)} = 121.625$.

Then we can decompose the user's response time constraints for each task by $C_i^{(r)} = C^{(r)}\frac{\mu_i^{(r)}}{\sum_{i=1}^{n}\mu_i^{(r)}}$, where $C^{(r)}$ represents the user's total constraints on the response time and n is the total number of tasks. Therefore, $C_i^{(r)}$ is the excepted value of the response time constraint for task t_i. The excepted value of the price constraint for task t_i, i.e., $C_i^{(c)}$, can be calculated in the same way. Note that the decomposition of constraints is only for service level setting, and during the selection of component service we take the constraints of composite service as a whole.

Here, taking the response time constraint as an example, we formulate the four state levels as formula (2), where $v_{ij}^{(r)}$ is the response time of service ws_{ij}. If $v_{ij}^{(r)}$ is less than or equal to $0.85C_i^{(r)}$, then the state level is 1; if $v_{ij}^{(r)}$ is more than $0.85C_i^{(r)}$ and less than or equal to $C_i^{(r)}$, then the state level is 2; if $v_{ij}^{(r)}$ is more than $C_i^{(r)}$ and less than or equal to the user's total constraints vector $C^{(r)}$, then the state level is 3; otherwise, if $v_{ij}^{(r)}$ is more than $C^{(r)}$ or the service fails, then the state level is 4. This can be denoted as

$$s_i^{(r)} = \begin{cases} 1 & v_{ij}^{(r)} \le 0.85C_i^{(r)} \\ 2 & 0.85C_i^{(r)} < v_{ij}^{(r)} \le C_i^{(k)} \\ 3 & C_i^{(k)} < v_{ij}^{(k)} \le C^{(k)} \\ 4 & v_{ij}^{(k)} > C^{(k)} \text{ or service failure,} \end{cases} \tag{2}$$

where 0.85 is a parameter to tune the levels. In this paper the four state levels for price are set in the same way. In fact, the number of levels and the scope of each level can be set up according to the actual situation.

3.3 Solution

The Q-learning algorithm is used to study the performances of each candidate service from the historical execution records. In this paper, we set the reward function as $r = 10(s - s') + 10$. This means that the reward is determined by the difference between s and s'. That is, the more the new state s' is superior to the old state s, the greater the reward is, and vice versa. If the new state s' is the same as the old state s, then the running service can obtain a basic reward. When the new state s' is much worse, i.e., s' is larger than s by 2 or more levels, then the reward is a negative value; in this case, the reward becomes a penalty.

Executing Algorithm 2 for the above example, the results of Q-learning are shown in the following matrix:

$$Q = \begin{pmatrix} 4.0219 & 1.1104 & 12.3300 & -0.0280 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1.2607 & 3.2561 & 6.7946 & 0 \\ 11.3940 & -7.0000 & 2.3327 & 0 \\ 15.5230 & 3.0000 & 25.3450 & 0 \\ 4.0790 & 12.8200 & 8.2120 & 8.4224 \\ 17.0120 & 19.1020 & 18.6990 & 21.5350 \\ 3.2118 & 14.6410 & 14.8750 & 9.6686 \end{pmatrix}.$$

The matrix Q indicates the performance scores of each service in different sustainable states (each task has three sustainable states, i.e., s=1, 2 or 3) in terms of the services' adaptability to different constraint situations. For example, the 5th row of matrix Q shows the performance scores of services in A(2) at state $s = 2$. From the data in this row, we can infer that service ws_{21} is the best choice for task t_2 at state $s = 2$. It is reasonable to suppose that the service composition always begins from a good start, so the initial state is always $s = 1$; thus, the elements of rows 2 and 3 in matrix Q are always zeros. Furthermore, because task t_2 has 3 candidate services, the elements of the 4th column at rows 4, 5, and 6 are all zeros. During service composition, matrix Q is the basis of service selection under different constraint satisfaction states.

After Q-learning, service composition can be carried out based on the matrix Q. Table 1 shows an execution of the service composition, from Table 1 we can see the selected service for each task, its actual QoS values in the execution, and the new constraints state after the execution. Due to the fact that service compositions always start from a good state, the service with the highest score in the 1st row of the matrix Q, i.e., ws_{13}, is selected for task t_1. Executing service ws_{13}, the state becomes $s = 2$. Hence, for task t_2, the best service is sought in state $s = 2$; that is, the service with the highest score in the 5th row, i.e., ws_{21}, is selected. After the execution of service ws_{21}, the state is still $s = 2$. So, based on the data of the 8th rows in matrix Q, service ws_{34} is selected and executed, and the state becomes $s = 1$. At this point, the service composition for the example is executed successfully. It is important to note that the QoS data should be recorded in the historical execution QoS records after the execution of each service.

As can be seen from Table 1, CSSC-MDP always looks for the best service in the current state of the matrix Q to execute. However, because of the dynamic nature of the Internet environment, the performance of service is uncertain. The following sections show the adaptivity of CSSC-MDP to uncertain service behaviors.

3.4 Adaptivity to Variable QoS

Dynamic environment leads to variable QoS values. Comparing Tables 1 and 2. In the running of Table 2, service ws_{21} takes a much longer response time in Table 2 than in

TABLE 2
Service Composition When QoS Changes

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	310	70	2
2	ws_{21}	1,521	140	3
3	ws_{33}	101	25	1

Table 1, and the state becomes $s = 3$. Therefore, based on the last row of the matrix Q, service ws_{33} are selected and executed.

From the historical execution records we will find the service ws_{33} usually has a shorter response time and a bit higher price. The results show the self-adaptivity of the CSSC-MDP to the variable QoS.

3.5 Adaptivity to Uncertain Service Behaviors

Because of network fault or service evolution, sometimes service failures may occur. Such as the running of the service composition in Table 3, service ws_{13}, the best candidate service for task t_1, encounters problems. Based on the matrix Q, service ws_{11} is the next-best service in the candidate service set A(1); therefore, service ws_{11} is selected as the alternative to service ws_{13}. After the execution of service ws_{11}, the state becomes $s = 2$. The best candidate service for t_2 in this state, with the maximum value in the 6th row of matrix Q, i.e., ws_{23}, is selected and executed, and the service composition can proceed. It shows when service failure occurrence CSSC-MDP can find a suitable alternative and compliment the service composition.

If the selected alternative service has not fulfilled task t_1 yet, the situation is as shown in Table 4. When service ws_{13}, following ws_{11}, also encounters failure, service ws_{12} is the best choice in the current situation. Therefore, service ws_{12} is executed to fulfill task t_1, and the service composition is able to continue.

In fact, as long as there exist qualified services for each task, service composition can be successfully completed. This shows the robustness of the CSSC-MDP.

3.6 Adaptivity to Different Constraints

Different users have different QoS constraints on the same functional requirement. Suppose, in this example the constraints are changed to

$$(responsetime \leq 1830) \text{ and } (price \leq 330). \tag{3}$$

Compared with the QoS constraints (1), the constraint on price is looser and the constraint on response time is tighter. In this case, the results of Q-learning are shown in the following matrix Q_1.

TABLE 3
Service Composition When Service Fails

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	-1	70	4
1	ws_{11}	417	82	3
2	ws_{23}	1,240	180	3
3	ws_{33}	109	25	2

TABLE 4
Service Composition When Continues Services Fail

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	-1	20	4
1	ws_{11}	-1	20	4
1	ws_{12}	289	95	3
2	ws_{23}	1,240	180	3
3	ws_{33}	130	25	3

$$Q_1 = \begin{pmatrix} -4.1436 & -7 & 11.941 & 2.9984 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -6.0003 & -7 & -13.654 & 0 \\ 6.1325 & 21.39 & 5.3209 & 0 \\ 19.652 & 22.403 & 20.629 & 0 \\ 0 & 0 & 0 & 0 \\ 16.943 & 22.996 & 17.753 & 12.674 \\ 9.8305 & 13.53 & 28.287 & 9.5256 \end{pmatrix}.$$

Comparing the 5th row of matrix Q_1 and matrix Q, we can see that service ws_{22} is the worst service for task t_2 in state $s = 2$ in Q, but in Q_1 service ws_{22} is the best service for task t_2 in the same state. This shows that service ws_{22} is more adaptive in circumstance of shorter response time and higher price. This is in accordance with the fact that service ws_{22} always has a lower mean response time and a higher price, which can be seen from the historical execution records. Based on matrix Q_1, the best composite service $(ws_{13}, ws_{22}, ws_{33})$ for the user constraints is performed successfully, as shown in Table 5.

If the QoS constraints of the composite service are changed to

$$(responsetime \leq 2700) \text{ and } (price \leq 255). \tag{4}$$

Compared with the constraints (1) and (3), the constraint on price is tighter and the constraint on response time is looser. In this case, the results of Q-learning are shown in the following matrix Q_2.

$$Q_2 = \begin{pmatrix} -10.2390 & -6.9993 & 2.7579 & -10.8440 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 19.8790 & -6.9322 & 11.5320 & 0 \\ 14.6440 & 2.9951 & 2.9988 & 0 \\ 0 & 0 & 0 & 0 \\ 20.1350 & 7.8807 & 11.448 & 7.8207 \\ 2.8271 & 2.7529 & 2.2797 & 1.5300 \end{pmatrix}.$$

Focusing on task t_3 in the state $s = 3$, the best service is service ws_{33} in Q; however, in Q_2 service ws_{31} is the best. This is because service ws_{31} is commonly lower in price and

TABLE 5
Service Composition with Constraint (3)

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	380	70	2
2	ws_{22}	1,021	230	3
3	ws_{33}	132	25	2

TABLE 6
Service Composition with Constraint (4)

Task No.	Selected service	Response time(s)	Price(yuan)	State
1	ws_{13}	373	70	3
2	ws_{21}	1,641	140	3
3	ws_{31}	146	20	1

higher in response time than service ws_{33} is, so ws_{31} is more adaptive to this circumstance. Hence, the running of composite service in this case is as shown in Table 6.

The results in Tables 5 and 6 show the adaptivity of CSSC-MDP to different user constraints.

4 Experimental Results and Discussion

To test the performance of CSSC-MDP on a larger scale, the following experiments simulate sequential service compositions which contain 20 tasks, and 50 candidate services per task. For each service, 100 historical response time records and a price data were artificially synthesized. The data set was constructed as follows: randomly generate a 1×20 vector $\boldsymbol{tpm}$ whose elements are between $[100, 1000]$; and randomly generate a 1×20 vector $\boldsymbol{tpv}$ whose elements are between $[10, 30]$; take tpm_i as the mean and tpv_i as the variance, randomly generate a 50×1 normal distribution data sp_i; $\boldsymbol{sp} = (sp_1, sp_2, , sp_{20})$ are 50×20 price data for the 20 tasks' 50 candidate services; let $\boldsymbol{trm} = [0.25 \times \boldsymbol{tpm}] \times 10$ are the mean response time of each task, the function is set to guarantee that for different tasks the longer average response time the higher price; randomly generate a 1×20 vector trv,whose elements are between $[100, 1000]$; take trm_i as the mean and trv_i as the variance, randomly generate 50×1 normal distribution data $\boldsymbol{srm_i}$ as the mean response time of 50 candidate services for task t_i; sort the elements in $\boldsymbol{srm_i}$, let a higher price corresponds to a shorter mean response time, it is in line with the actual situation that for the same task the shorter response time the higher price; randomly generate a 50×20 vector $\boldsymbol{srv}$ whose elements are between $[50, 400]$; take srm_{ij} as the mean and srv_{ij} as the variance, randomly generate a 100×1 normal distribution data $\boldsymbol{srr_{ij}}$, $\boldsymbol{srr}$ are $100 \times 50 \times 20$ historical recorded response time data for the 20 tasks' 50 candidate services 100 records.

The simulations were conducted by using MATLAB R2013a. The experimental platform runs Windows 10 with an Intel Core Quad CPU at a clock speed of 2.67 GHz with 4 GB RAM.

We assume that the service composition system has no knowledge of the QoS performance of all the candidate services. Based on the historical execution QoS records, we let the Q-learning algorithm guide the service composition to reach the optimal policy gradually. In the following experiments, the parameters of Q-learning are determined first; then, the efficiency of Q-learning is studied; finally, comparison experiments with three existing methods are conducted.

4.1 Parameter Tuning

The learning rate and the greedy rate are two important parameters in the Q-learning algorithm. The following experiments show the determination of the two parameters.

4.1.1 Learning Rate

To obtain a faster learning speed and a better learning result, it is necessary to set a proper learning rate. We fix the number of tasks to 5 and the number of candidate services of each task to 10, and vary the learning rate α. Fig. 4 shows the learning efficiency of Q-learning with learning rate $\alpha = 0.1, 0.3, 0.5$. The learning speed here refers to the number of iterations after which the cumulative rewards achieves the maximum. From Fig. 4 we can see, when $\alpha = 0.1$, the cumulative rewards still have an increasing trend after about 100,000 iterations (see Fig. 4a), which indicates the learning speed is low. As demonstrated in Figs. 4b and 4c, when $\alpha = 0.3$, the cumulative rewards reach the maximum before 100,000 iterations; when $\alpha = 0.5$, the cumulative rewards reach the maximum before 40,000 iterations; the cumulative rewards reach quickly to a higher value indicates the learning speeds are higher. However, we can also see from Figs. 4b and 4c the cumulative rewards fluctuate more and more severely.

It is reasonable that the smaller the learning rate, the slower the learning speed, whereas a higher learning rate results in a severe fluctuation range. Hence, we take a dynamic learning rate, initially set $\alpha = 0.5$, and consider that every iteration has an $\alpha = \alpha - 0.000007$ decay. Fig. 5 shows learning efficiency with this dynamic learning rate. From Fig. 5, we can see cumulative rewards reach to a higher value after 20,000 iterations, and the fluctuations of cumulative rewards are gradually mild. Hence, we use this dynamic learning rate in the following experiments.

4.1.2 Greedy Rate

To study the influence of the greedy rate on Q-learning, we vary the parameter ε of ε-greedy algorithm from 0.1 to 0.9.

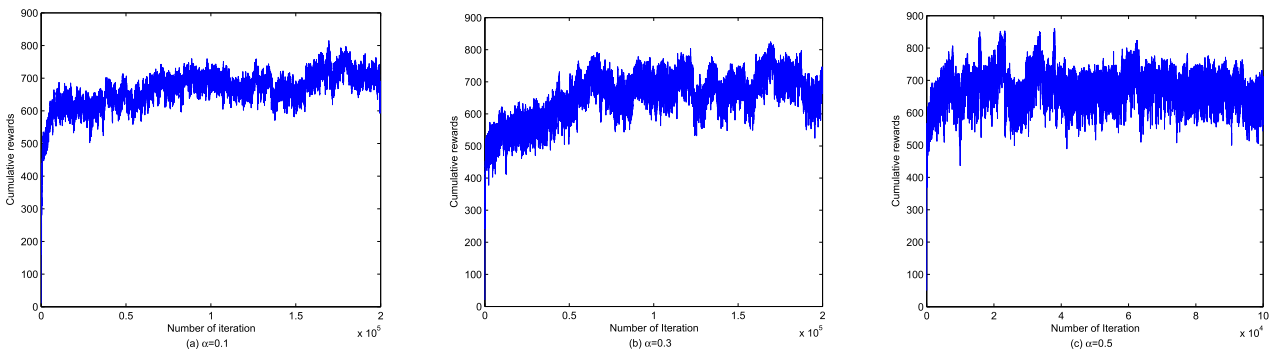


Fig. 4. Efficiency of Q-learning with different learning rates.

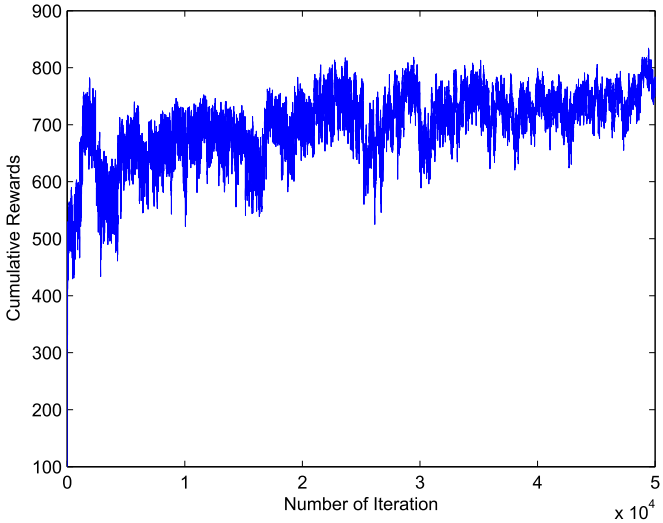


Fig. 5. Q-learning efficiency of a dynamic learning rate.

Fig. 6 shows the convergent cumulative rewards and the number of iterations with different ε values.

As demonstrated in Fig. 6, the iteration number in Q-learning reaches the minimum when the ε-greedy rate is $\varepsilon = 0.6$, and in this case the cumulative reward convergence is near-optimal. Hence, we set the ε-greedy rate of $\varepsilon = 0.6$ in the following experiments.

4.2 Performance Study

In the experiments shown in Fig. 7, we study the performance with respect to the number of tasks and the number of candidate services. The number of tasks varies from 5 to 20, while the number of candidate services of each task varies from 5 to 20. We take moderate constraints for each case in order to ensure that all the 3 different states can be reached in every case. In each case, repeat the experiments 100 times, and record the average learning time.

Fig. 7 illustrates that the time of Q-learning increases obviously with the number of tasks. It is easy to understand, because one more task in the work-flow brings one more step in each iteration of the Q-learning. Fig. 7 also illustrates that it is not obvious the time of Q-learning increases with

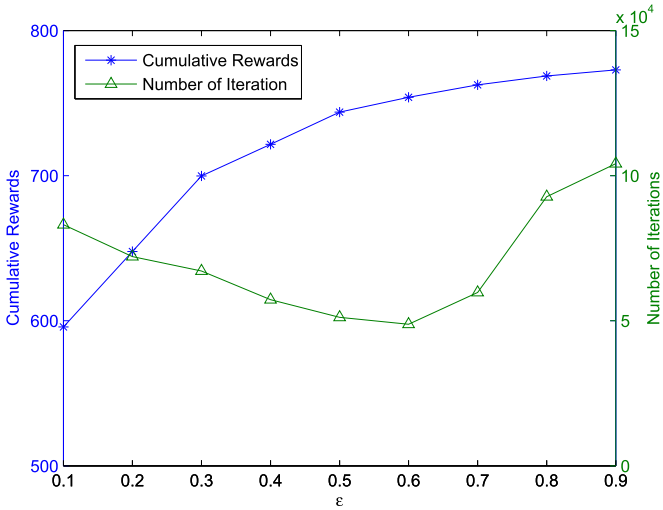


Fig. 6. Q-learning efficiency of different greedy rates.

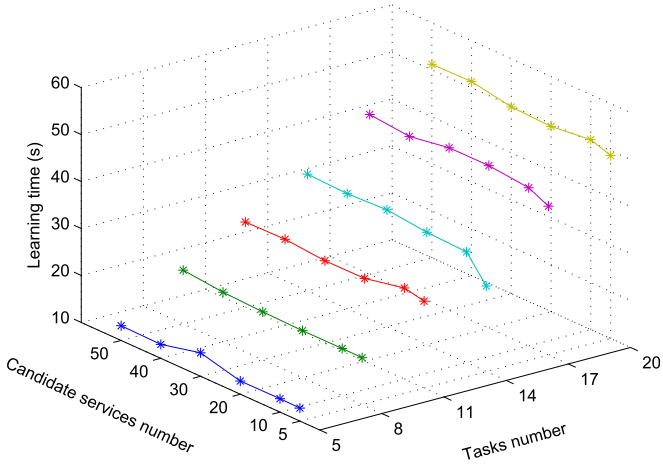


Fig. 7. Performance of Q-learning.

the number of candidate services, this is because the time cost of Q-learning depends not only on the number of candidate services but also on the diversity of candidate services.

4.3 Comparison Experiments

For the purpose of performance evaluation, we compare the method proposed in this paper with a classic global optimization method *WS-IP* [5], a local optimization method *Hybrid* proposed in [13] and an MDP-based method [23], for the sake of convenience, we name it *RQASC*.

WS-IP. The purpose of this method is to select the candidate service for each task that satisfies the user's constraint and maximizes a defined utility function. In this method, the CSSC problem is modeled as a 0-1 integer programming (IP) problem and the well-known *lpx-intopt* algorithm is used to find the optimal solution. In [13] and [5], *WS-IP* is regarded as the optimal solution to be compared.

Hybrid. This is a method based on the decomposition of constraints. This method first uses a mixed-integer program (MIP) to find the optimal decomposition of the global constraints, and then find the best services that satisfy the local constraints. The purpose of decomposition is to improve the optimization efficiency.

RQASC. Being aware of the undetermined QoS, this method measures the QoS by the mean and variance of random variables, and model the service composition as an MDP. RQASC only selects the optimal composite service, and it cares nothing about user's QoS constrains.

To simulate the uncertain environments, 2,326 service failure records were randomly inserted into the response time record data. That is, in our artificially synthesized database, the service failure rate was 0.02326 and we set the constraints as $(Price \leq 7500)$ and $(ResponseTime \leq 20000)$.

Without loss of generality, suppose that the response time and the price are of equal importance; hence, in the WS-IP and hybrid methods, the weight of both response time and price are set to 0.5. In RQASC method, the price and the response time are alternately optimized to obtain the highest success rate. For each method mentioned above, we conduct service composition 100 times. The average success rate of WS-IP is 54 percent, the average success rate of Hybrid is 31 percent, the average success rate of RQASC is 72 percent, and the average success rate of CSSC-MDP is

TABLE 7
Response Time and Price of a Successful Execution

Task No.	Response Time (s)				Price (Yuan)			
	WS-IP	Hybrid	RQASC	CSSC-MDP	WS-IP	Hybrid	RQASC	CSSC-MDP
1	**498.94**	512.02	545.68	537.40	215.75	178.33	**163.44**	**163.44**
2	**716.22**	735.90	746.12	746.48	271.80	257.33	**231.34**	**231.34**
3	283.61	313.18	**275.60**	315.29	134.59	106.19	145.52	**92.33**
4	**463.90**	476.26	465.49	465.36	178.68	**157.43**	220.64	**157.43**
5	**919.64**	942.00	936.69	927.12	342.36	339.46	**317.70**	**317.70**
6	1,518.34	1,554.24	**1,438.85**	1,503.03	579.77	**575.31**	661.16	589.92
7	**566.48**	632.37	585.25	645.41	255.89	223.95	268.44	**202.44**
8	421.91	423.92	423.20	**418.20**	149.72	**137.05**	187.65	137.17
9	**1,164.50**	1,228.97	1,237.91	1,171.89	503.66	459.36	**440.37**	493.72
10	389.11	374.05	374.53	**370.64**	113.65	125.53	**107.02**	118.96
11	2,004.08	1,981.03	**1,939.09**	2,019.56	772.18	**766.89**	862.04	781.65
12	1,735.82	1,765.65	1,739.19	**1,703.49**	643.36	648.06	**626.83**	643.17
13	**468.03**	507.36	510.17	510.25	200.81	175.97	**157.24**	163.95
14	584.97	583.46	**492.46**	564.53	213.48	204.85	267.93	**191.69**
15	**648.18**	653.73	674.88	685.28	239.27	229.67	**207.72**	219.46
16	1,636.47	1,624.78	**1,518.87**	1,611.44	626.32	**604.97**	717.74	633.16
17	813.01	839.17	**758.23**	838.11	313.01	301.80	363.65	**277.11**
18	**773.12**	830.81	803.65	815.80	325.43	**276.48**	**276.48**	**276.48**
19	**940.74**	981.64	1,043.04	1,005.90	418.65	349.34	**314.89**	337.41
20	**1,916.04**	1,990.62	1,982.29	1,959.30	757.84	**684.12**	**684.12**	690.02
Total	**18,463.11**	18,951.16	18,491.23	18,814.48	7,256.22	6,802.09	7,108.14	**6,718.55**

100 percent. From the result we can see the poor adaptivity of WS-IP and Hybrid to dynamic environments. Once one of the selected optimal candidate services fails, the service composition will inevitably face failure. The success rate of RQASC is a little higher. This is because it can guarantee the success of service composition but the QoS constrains are ignored by RQASC. In contrast, CSSC-MDP can perceive the state of service composition and always selects the service best suited to the situation. Moreover, in the case of component service failure, CSSC-MDP can select another qualified candidate service to fulfill the task. Therefore, even if the environment is dynamic, as long as there are qualified candidate services, CSSC-MDP can manage to complement the service composition successfully.

Table 7 shows the response time and the price records recorded from one successful running of the 4 methods.

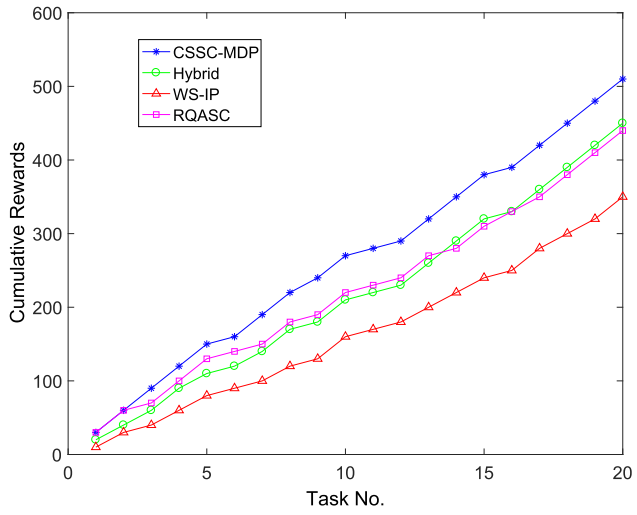


Fig. 8. Cumulative reward comparison for 4 methods.

From Table 7, we can see each method has the best performance for response time or price. On the whole, WS-IP is superior in response time, but the price of composition service by WS-IP is the most expensive. While CSSC-MDP performs the best in terms of price, and from the perspective of response time, CSSC-MDP performs the best for 3 tasks and approximates the optimal values for other tasks. This shows that CSSC-MDP can obtain the best tradeoff between the response time and the price.

To compare the degree of satisfaction to user constraints, Fig. 8 shows the cumulative rewards of the successful execution of the 4 methods. From Fig. 8, we can see CSSC-MDP has the highest cumulative rewards. This illustrates that CSSC-MDP is best adaptive to the variable QoS values, and satisfy the user constraints as well as possible.

4.4 Experiments on Real-World Dataset

To further evaluate the performance of CSSC-MDP, we do service composition experiments on the dataset 2 of WS-DREAM, a real-world QoS dataset released by [24] and [25]. The dataset includes response time data $rtmatrix$ and through-put data $tpmatrix$, which are collected from 339 users on 5,825 Web services. We group the services into 50 tasks, hence, each task has about 116 candidate services, and each service has 339 response time and through-put records.

Setting user constraints as $(ResponseTime \leq 22s)$ and $(ThroughPut \geq 13\ Kbps)$, and setting the number of tasks $t = 10, 20, 30, 40, 50$, we do the service composition using the 4 methods mentioned above for 1,000 times respectively. The results are shown in Table 8 and Fig. 9.

From the data in Table 8, we can see the success rate of CSSC-MDP on the real world dataset is significantly higher than that of other methods. More seriously, for methods WS-IP and Hybrid, when the number of tasks $t \geq 30$, the service compositions are almost all failed. This is because

TABLE 8
Comparison of Times of Successful Execution

Method	t=10	t=20	t=30	t=40	t=50
CSSC-MDP	795	615	514	445	276
RQASC	174	45	37	29	16
WS-IP	249	19	4	1	0
Hybrid	394	17	3	0	0

the uncertainty of service composition increases rapidly as the number of tasks increases. Fortunately, CSSC-MDP is more adaptive to the variable QoS and service behavior, so CSSC-MDP is relatively the most robust method.

Fig. 9 shows the mean cumulative reward of the successful executions of the 4 service composition methods. From Fig. 9 we can see CSSC-MDP is obviously superior to the others when $t = 10, 20, 40, 50$. And when $t = 30$, CSSC-MDP is very close to the highest cumulative reward. In conclusion, experimental results on the real world dataset verify the superior adaptivity of CSSC-MDP to the uncertain QoS and service behavior.

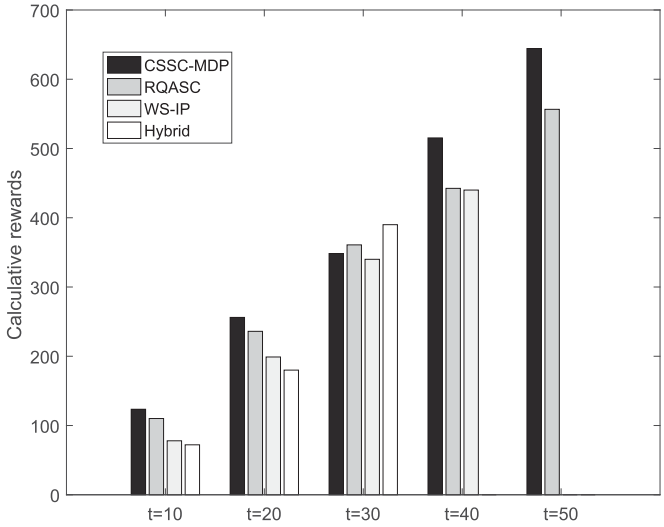


Fig. 9. Comparison of cumulative rewards on a real dataset.

5 Related Work

In this section, we study the existing methods for constraint-satisfied service composition at first, then we discuss the service composition methods using reinforcement learning.

5.1 Methods for CSSC

The existing methods for solving CSSC problem can roughly be grouped into two groups: global optimization and local optimization.

5.1.1 Global Optimization Methods

The global optimization methods attempt to find the optimal composite service among all possible combinations while meeting user constraints.

In one of the early studies on constraint-satisfied service composition, Hassine et al. [4] formalized the CSSC problem as a constraint optimization problem at first; then an incremental user-intervention-based protocol was used to find the optimal composite service at run time. Yu et al. [5] modeled the CSSC problem in two ways: a multi-dimension multi-choice knapsack problem (MMKP) model and a multi-constrained problem (MCOP) model, and for each model, an efficient heuristic algorithm was proposed. Zhao et al. [11] modeled the CSSC problem using the weighted Tchebycheff distance, avoiding the limitations of linear functions in setting the weight of the QoS attributes, and proposed two evolutionary algorithms to solve optimal problems in different scenarios. Lecue and Mehandjiev [26] balanced semantic fit with QoS metrics, modeled the CSSC problem as a constraint-satisfaction problem, and adapted a hill-climbing algorithm to compute a "good enough" solution that met initial constraints rather than computing the optimal composition. Caporuscio et al. [27] built both design-time and run-time models for the service composition and identified the service composition satisfying the QoS requirements. The quality attributes of the selected composition are monitored, analyzed and, if necessary, plans are generated in terms of modifications.

Other optimization methods, such as, Ardagna and Pernici [6], Kritikos and Plexousakis [7] and He et al. [9] consider the CSSC process as mixed-integer linear programming (MILP) problem, and Garcia et al. [10] model the CSSC problem as constraint shortest path problem. In theory, such exhaustive global optimization methods can get the global optimal constraint-satisfied composite service, if it exists. However, the efficiency of global optimization methods decreases dramatically as the problem grows. Some approximate optimization algorithms, such as the stochastic search method [26], particle swarm optimization [28], evolutionary algorithm [11] and so on, have been studied to improve optimization efficiency in the CSSC problem. In general, most of the global service composition methods assume the QoS values are fixed and the service behaviors are determined, as a result, their adaptivity to the dynamic environments are poor.

5.1.2 Local Optimization Methods

The local optimization method decomposes global constraints for tasks in the workflow and selects the optimal service for each task, so as to meet local constraints independently.

Sun et al. [12] computed the utility of a composite service from the utilities of component services and derived the constraints of component services from the constraints of the composite service. Alrifai et al. [13] used a global optimization method to find the optimal decomposition of constraints, and then used the distributed local selection to find the best services satisfying the local constraints. Raj and Sasipraba [14] used local constraints as the thresholds to filter unqualified services and took service utility as the key value to select the optimal service for a task. Then the highest-ranked service was provided to user.

Because of the undetermined QoS values, it is almost impossible to get an appropriate decomposition of global constraints. The local constraints are either too strict or too loose, and the success rate of service composition is inevitably reduced.

Table 9 shows the pros and cons of these methods.

TABLE 9
Methods for CSSC

Method/Model	Advantage	Weakness
incremental user-intervention-based protocol [4]	adaptive to stochastic service evolution	not adaptive to the variable QoS
MMKP and MCOP [5]	optimize the QoS of the composite service	not adaptive to the variable QoS
MILP [6], [7], [9]	get the global optimal solution	the efficiency decreases dramatically as the problem grows
multi-objective optimization [11]	the limitations of linear functions are avoided	not adaptive to the variable QoS
hill-climbing algorithm [26]	high solving efficiency	the solution is locally optimal and not adaptive to the dynamic environment
design-time model and run-time model [27]	services that most likely contribute in QoS violations are get rid off	new services have no chance to take part in
local selection approach[12]	high efficiency	not adaptive to the variable QoS
constraints decomposition [13]	the decomposition of constraints is optimized	QoS values are regarded as determined
local-global combined [14]	use local constraints to filter unqualified services	not adaptive to the variable QoS and uncertain service behaviors

TABLE 10
Service Composition Methods Using Reinforcement Learning

Model	Advantage	Weakness
MDP+Bayesian learning [17]	can generate robust workflow	can not guarantee the optimality of composite services
MDP+Bayesian learning [29]	improve the quality of Workflow through Bayes learning	without considering the QoS of the composite service
MDP [30]	different structures of service composition are considered	only one QoS attribute can be optimized
MDP+HTN planning [31]	multiple QoS attributes are taken into consideration	take the QoS values as be determined
MDP[23]	measure QoS by mean and variance, reduce the probability of composite service failure	the real QoS value could not be consistent with the theory distribution
improved MDP [32]	facilitate the service composition by improving the optimization equation	the real QoS value could not be consistent with the theory distribution
MDP [33]	integrate multiple workflows and alternative services into service composition	the optimal service composition can only be find in the long run
team Markov Games [34]	applicable to the distributed environment	the optimal service composition can only be find in the long run
MDP [35]	provide flexible service composition	can only get the near optimal composite service

5.2 Service Compositions by Reinforcement Learning

Reinforcement learning is a typical technology used for planning and optimization in dynamic environments, and many researches has used reinforcement learning to conduct service composition, such as [17] considering the undermined service behaviors, [29] considering the uncertainty of the acture environment, [30] discussing the optimization of different composition structure, [31] finding multiple composition plans and selecting the most appropriate for user, [23] concerning the undetermined QoS, [32] facilitating the service composition, [33] considering the indeterminacy of service behavior, [36] finding the optimal workflow, [34] improving the optimization efficiency for large-scale service composition, [37] presenting an approach of multi-agent service composition, [35] building service pair to handle the change in dynamic environment etc. Table 10 shows the pros and cons of some presentative methods.

In contrast to these works, the proposed CSSC-MDP satisfies the user's functional requirements and QoS constraints furthest and considers the conflicts between the QoS attributes. Moreover, CSSC-MDP selects a component service after the execution of the previous service. Thus, before the selection of a component service, the execution QoS of the previous services are certain values, hence the constraints which need to be satisfied by the following services can be calculated. Based on the results of calculation, the component service mostly satisfied the constraints can be selected. Even if in the case where a selected service failed, the optimal alternative service can be selected immediately to replace it. Therefore, CSSC-MDP is highly adaptive to the dynamic and uncertain environment.

6 Conclusions

In this paper, we presented a reinforcement learning method for solving the CSSC problem in the dynamic environment. This method built a CSSC-MDP model based on an MDP model and used a Q-learning algorithm to solve the model. Extensive experiments show a significant improvement in terms of adaptivity to the uncertain behavior of services in the dynamic environment. In comparison with a global optimization method, a local optimization method and an MDP-based service composition method, CSSC-MDP also showed the superiority in terms of the satisfaction of user's QoS constraints.

The main contributions of this paper are threefold: first, considering the uncertain behavior of service, we selected component services during execution of the service composition; this avoids the service composition failure brought by the failure of a component service. Second, considering that QoS attributes are not always the same for different executions, we select a component service based on the real execution QoS data of the previous services; this will help to calculate the real constraints that need to be satisfied by the following services. Third, the selection strategy of Q-learning is globally optimized, this guarantees the optimality of the composite service.

Acknowledgments

The work described in this paper was partially supported by the National Natural Science Foundation of China(No. 61673249,61273291), Research Project Supported by Shan xi Scholar ship Council of China(No. 2016-004).

References

[1] I. Ari and N. Muhtaroglu, "Design and implementation of a cloud computing service for finite element analysis," *Advances Eng. Softw.*, vol. 60, pp. 122–135, 2013.

[2] R. Rezaei, T. K. Chiew, S. P. Lee, and Z. S. Aliee, "A semantic interoperability framework for software as a service systems in cloud computing environments," *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5751–5770, 2014.

[3] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decades overview," *Inform. Sci.*, vol. 280, pp. 218–238, 2014.

[4] A. B. Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal web service composition," in *Proc. Int. Semantic Web Conf.*, 2006, pp. 130–143.

[5] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end QoS constraints," *ACM Trans. Web*, vol. 1, no. 1, 2007, Art. no. 6.

[6] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, Jun. 2007.

[7] K. Kritikos and D. Plexousakis, "Mixed-integer programming for QoS-based web service matchmaking," *IEEE Trans. Serv. Comput.*, vol. 2, no. 2, pp. 122–139, Apr.-Jun. 2009.

[8] D. Ardagna and B. Pernici, "Global and local QoS constraints guarantee in web service selection," in *Proc. IEEE Int. Conf. Web Serv.*, 2005, pp. 805–806.

[9] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction," *IEEE Trans. Softw. Eng.*, vol. 40, no. 2, pp. 192–215, Feb. 2014.

[10] G. A. Garcia Llinas and R. Nagi, "Network and qos-based selection of complementary services," *IEEE Trans. Serv. Comput.*, vol. 8, no. 1, pp. 79–91, Jan.-Feb. 2015.

[11] X. Zhao, L. Shen, X. Peng, and W. Zhao, "Toward SLA-constrained service composition: An approach based on a fuzzy linguistic preference model and an evolutionary algorithm," *Inform. Sci.*, vol. 316, pp. 370–396, 2015.

[12] S. X. Sun and J. Zhao, "A decomposition-based approach for service composition with global QoS guarantees," *Inform. Sci.*, vol. 199, pp. 138–153, 2012.

[13] M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient web service composition with end-to-end QoS constraints," *ACM Trans. Web*, vol. 6, no. 2, 2012, Art. no. 7.

[14] R. J. R. Raj and T. Sasipraba, "Web service selection based on qos constraints," in *Proc. IEEE Trendz Inf. Sci. Comput.*, 2010, pp. 156–162.

[15] Q. Hu and J. Liu, "An introduction to markov decision processes," Xidian University, Xian, China, 2000.

[16] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Sci., Serv. Agents World Wide Web*, vol. 1, no. 3, pp. 281–308, 2004.

[17] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using Markov decision processes," in *Proc. IEEE Int. Conf. Web Serv.*, 2004, pp. 576–582.

[18] A. Slomiski, "On using BPEL extensibility to implement OGSI and WSRF grid workflows," *Concurrency Comput.: Practice Experience*, vol. 18, no. 10, pp. 1229–1241, 2006.

[19] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, "Clustering web services to facilitate service discovery," *Knowl. Inf. Syst.*, vol. 38, no. 1, pp. 207–229, 2014.

[20] W. Chen, I. Paik, and P. C. Hung, "Constructing a global social service network for better quality of web service discovery," *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 284–298, Mar.-Apr. 2015.

[21] E. Al-Masri and Q. H. Mahmoud, "Qos-based discovery and ranking of web services," pp. 529–534, 2007.

[22] R. Bellman, "A markovian decision process," DTIC Document, 1957.

[23] X. Fan, C. Jiang, J. Wang, and S. Pang, "Random-qos-aware reliable web service composition," *J. Softw.*, vol. 20, no. 3, pp. 546–556, 2009.

[24] Z. Zheng, Y. Zhang, and M. R. Lyu, "Distributed qos evaluation for real-world web services," in *Proc. IEEE Int. Conf. Web Serv.*, 2010, pp. 83–90.

[25] Y. Zhang, Z. Zheng, and M. R. Lyu, "Exploring latent features for memory-based QoS prediction in cloud computing," in *Proc. 30th IEEE Symp. Reliable Distrib. Syst.*, 2011, pp. 1–10.

[26] F. Lecue and N. Mehandjiev, "Satisfying end user constraints in service composition by applying stochastic search methods," *Web Service Composition New Frameworks Designing Semantics: Innovations: Innovations*, 2012, Art. no. 238.

[27] M. Caporuscio, R. Mirandola, and C. Trubiani, "Qos-based feedback for service compositions," in *Proc. 11th Int. ACM SIGSOFT Conf. Quality Softw. Archit.*, 2015, pp. 37–42.

[28] X.-Q. Fan, "A decision-making method for personalized composite service," *Expert Syst. Appl.*, vol. 40, no. 15, pp. 5804–5810, 2013.

[29] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic workflow composition using Markov decision processes," *Int. J. Web Serv. Res.*, vol. 2, no. 1, 2005, Art. no. 1.

[30] A. Gao, D. Yang, S. Tang, and M. Zhang, "Web service composition using Markov decision processes," in *Advances in Web-Age Information Management*. Berlin, Germany: Springer, 2005, pp. 308–319.

[31] J. Xu, K. Chen, and S. Reiff-Marganiec , "Using Markov decision process model with logic scoring of preference model to optimize htn web services composition," *Int. J. Web Serv. Res.*, vol. 8, no. 2, pp. 53–73, 2011.

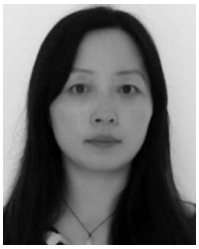
[32] X. Fan, X. Fang, and Z. Ding, "Indeterminacy-aware service selection for reliable service composition," *Frontiers Comput. Sci. China*, vol. 5, no. 1, pp. 26–36, 2011.

[33] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya, "Adaptive service composition based on reinforcement learning," in *Service-Oriented Computing*. Berlin, Germany: Springer, 2010, pp. 92–107.

[34] H. Wang, Q. Wu, X. Chen, Q. Yu, Z. Zheng, and A. Bouguettaya, "Adaptive and dynamic service composition via multi-agent reinforcement learning," in *Proc. IEEE Int. Conf. Web Serv.*, 2014, pp. 447–454.

[35] J. Yang, W. Lin, and W. Dou, "An adaptive service selection method for cross-cloud service composition," *Concurrency Comput.: Practice Experience*, vol. 25, no. 18, pp. 2435–2454, 2013.

[36] H. Wang, X. Zhou, X. Zhou, W. Liu, and W. Li, "Adaptive and dynamic service composition using q-learning," in *Proc. 22nd IEEE Int. Conf. Tools Artif. Intell.*, 2010, pp. 145–152.
[37] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, and A. Bouguettaya, "Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition," in *Service-Oriented Computing*. Berlin, Germany: Springer, 2014, pp. 154–168.



Lifang Ren received the MS degree from the Department of Computer Science and Technology, Taiyuan University of Science and Technology, in 2004. She is currently working toward the PhD degree in the School of Computer and Information Technology, Shanxi University. Now she is also a lecture in the School of Applied Mathematics, Shanxi University of Finance and Economics. Her main research interests include service computing and trustworthy software.

Wenjian Wang received the PhD degree from the Institute for Information and System Science, Xi'an Jiaotong University, in 2004. Now she is a full-time professor and PhD supervisor at School of Computer and Information Technology and Key Laboratory of Computational Intelligence and Chinese Information Processing, Shanxi University. She has published more than 70 academic papers. Her current research interests include machine learning, data mining, etc.

Hang Xu received the MS degree from School of Computer and Information Technology, Shanxi University, in 2014. She is currently working toward the PhD degree in the School of Computer and Information Technology, Shanxi University. Her main research interests include service computing and machine learning.

Customized Network Security for Cloud Service

Jin He, Kaoru Ota, *Member, IEEE*, Mianxiong Dong, *Member, IEEE*, Laurence T. Yang, *Senior Member, IEEE*, Mingyu Fan, Guangwei Wang, and Stephen S. Yau, *Life Fellow, IEEE*

Abstract—Modern cloud computing platforms based on virtual machine monitors (VMMs) host a variety of complex businesses which present many network security vulnerabilities. In order to protect network security for these businesses in cloud computing, nowadays, a number of middleboxes are deployed at front-end of cloud computing or parts of middleboxes are deployed in cloud computing. However, the former is leading to high cost and management complexity, and also lacking of network security protection between virtual machines while the latter does not effectively prevent network attacks from external traffic. To address the above-mentioned challenges, we introduce a novel customized network security for cloud service (CNS), which not only prevents attacks from external and internal traffic to ensure network security of services in cloud computing, but also affords customized network security service for cloud users. CNS is implemented by modifying the Xen hypervisor and proved by various experiments which showing the proposed solution can be directly applied to the extensive practical promotion in cloud computing.

Index Terms—Network security, FDCs, unified management, customized network security service, packet delay, throughput

1 Introduction

Cloud computing has emerged as one of the most influential paradigms in the IT industry, and has attracted extensive attention from both academia and industry. Reduced costs and capital expenditures, increased operational efficiencies, scalability, and flexibility are regarded as benefits of cloud computing. Although the great benefits brought by cloud computing paradigm are exciting for IT companies, academic researchers and potential cloud users, security problems of cloud computing become serious obstacles which, without being appropriately addressed, will limit extensive applications and utilization of cloud computing in the future. In cloud computing, network security [8], [9], [17], [46], [53] is believed to be one of the prominent security concerns, and it poses the same deadly threat as data security and privacy disclosure. Furthermore, as stated by National Vulnerability Database [29], there are 84 network vulnerabilities discovered in cloud computing by February 2013, all of which strongly threaten network security of cloud computing. In addition, there is sufficient evidence [27] that a large number of data destruction or tampering or forgery in cloud computing still come from malicious network attacks.

In recent years, there have been a number of relative efforts [1], [18], [22], [28], [47], [48], [52] in probing into data security and privacy in cloud computing, and tremendous progress has been maintained. However, these outcomes are based on an assumption that there has been secure network of cloud computing, and if the assumption is got rid of, the above achievements would come to be naught. Further, some researchers pay much attention to certain types of network security in cloud computing. For example, Lin et al. [19] have placed network inspection detection system into a privileged virtual machine (VM) to verify all packets received by the cloud platform. However, this approach has an unavoidable drawback: the privileged VM causes serious performance bottlenecks. Wu et al. [50] focus on the security of virtual network in virtualized environment and solve network security between VMs by Firewall. However, it is powerless for attacks from malicious external traffic. McAfee Security-as-a-Service [34] merely focuses on Email and Web protection in cloud computing, Imperva Cloud [41] and Du et al. [5] provide Distributed Denial-of-Service (DDoS) protection service, and Krishnan et al. [14] attach importance to intrusion detection system in cloud computing. Huawe security products [30] also only provides a single type of network security service for cloud computing. The preceding solutions are provided for a single type of service protection or detection in cloud computing (e.g., Web or E-mail), and they are lacking of integrated comprehensive protection for multi-service cloud.

Since cloud computing hosts multi-type network-based service which requires a desired sequence of multiple middleboxes together to protect their network security. For example, Web service needs Firewall and Web Application Firewall chains (FW-WAF) to protect network security.

- *J. He, M. Fan, and G. Wang are with School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, P.R. China. E-mail: hejin_some@163.com, ff98@uestc.edu.cn, gwwanguestc@hotmail.com.*
- *K. Ota and M. Dong are with Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran, Hokkaido 050-8585, Japan. E-mail: {ota, mx.dong}@csse.muroran-it.ac.jp.*
- *L. T. Yang is with School of Information and Communication Engineering, University of Electronic Science and Technology of China, China, and Department of Computer Science, St Francis Xavier University, Canada. E-mail: ltyang@gmail.com.*
- *S.S. Yau is with Computer Science and Engineering at Arizona State University, Tempe, AZ 85281. E-mail: yau@asu.edu.*

Manuscript received 23 Jan. 2016; revised 12 June 2016; accepted 8 July 2016. Date of publication 11 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Kaoru Ota.)
Digital Object Identifier no. 10.1109/TSC.2017.2725828

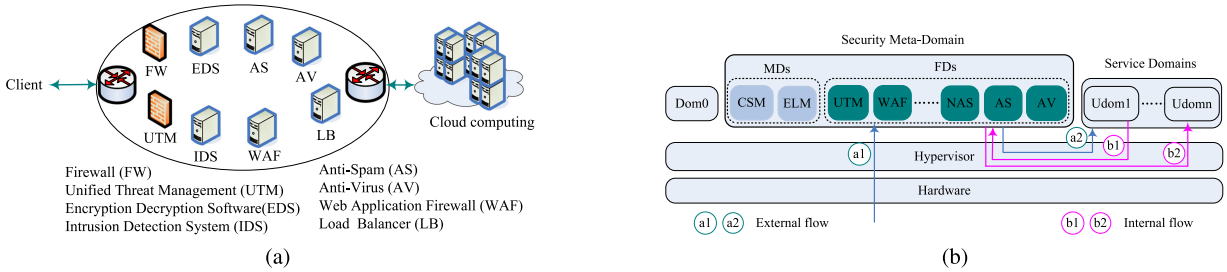


Fig. 1. Architecture comparison between the traditional architecture and CNS, (a) the traditional architecture, (b) CNS architecture, requiring external or internal traffic to traverse a desired sequence of FDs before accessing servers in service domains.

Thus, single network security service is unable to meet network security requirements for cloud computing. Considering the above shortcomings of single network security service, both industries and academies put many efforts on alternative solutions. In industry, traditional architecture [40] [10], [42] from Fig. 1a is regarded as current prevalent solution for multi-type service cloud, requiring large-scale security middleboxes, which leads to high costs [42], high complexity, and serious performance overhead. Besides, the architecture does not effectively prevent attacks between VMs [33], [48]. In academia, recent efforts [6], [13], [24] and [35] well combine middleboxes with SDN to protect enterprise network security and provide a flexible scalability and resource optimization for middleboxes. However, they lack of automatic security rules configuration and unified log management for middleboxes, and cannot provide appropriate cloud security.

Since above-mentioned efforts is inappropriate or defective to protect network security of cloud computing, the CNS system is presented that which adopts novel approach to eliminate or mitigate the disadvantages with promising benefits for cloud computing—reduced expenditure for infrastructure, personnel and management, pay-by-use, etc. As shown in Fig. 1b, the scheme is put forward in which security middleboxes are placed in cloud computing instead of at front-end of cloud computing so as to prevent malicious attacks from external and internal traffic. This will end mutual attacks between VMs for the traditional architecture. For security requirements in which cloud users' service is placed in cloud computing, CNS offers customized network security service to meet on-demand network security service. CNS also offers automatic security rules configuration and unified log management for middleboxes so as to lower complexity management and costs for cloud provider. Note that security capabilities or optimization algorithm of each device or middlebox [12] is not enhanced under this approach, but a more affordable and convenient protection service is provided.

In summary, our main contributions are as follows:

- *Innovative architecture* A novel flexible effiective architecture for network protection of cloud computing is proposed. Based on best knowledge, a systematic approach to provide on-demand unified solution for network security protection of cloud computing is advocated.
- *Preventing attacks from external and internal traffic* CNS prevents network attacks not only from external traffic but also attacks from internal traffic so as to ensure network security of cloud users' service.
- *Customized network security service* So long as cloud users understanding their service security hosted on cloud computing raise security requirements, CNS can provide network security protection for their service.
- *Low cost and complexity* CNS provides virtual middlebox with automatic security rules configuration, and offers unified log UI for a cloud user and cloud administrator. By this approach, cloud providers pay lower price to provide cloud users with safe and trusted security service. Accordingly, cloud users also have access to low-cost service fees.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 provides an overview of the CNS design. Section 4 gives implementation details of the entire system. Section 5 presents various experimental results for evaluating system impact and performance. The paper is concluded Section 6.

2 Related Work

This section presents literature review on several research areas related to CNS, including cloud-based single network security service and cloud-based integrated security service.

Cloud-based single security service. Focus on providing the security for a certain type of service, preventing certain types of attacks, or optimizing a certain type of middleboxes.

Cloud computing + IDS: In recent years, instruction detection system (IDS) for cloud computing has become research focus for numerous experts studies. For vulnerabilities of a cloud system and compromising virtual machines to deploy further large-scale DDoS, NICE [3] has proposed a multi-phase distributed vulnerability detection, measurement, and countermeasure selection mechanism, which is constructed on attack graph based analytical models and reconfigurable virtual network-based countermeasures to significantly improve attack detection and mitigate attack consequences. Because of distributed nature, grid and cloud computing environments can become the targets which intruders look for and become possible vulnerabilities to exploit. Meanwhile, it requires more than user authentication with passwords or digital certificates and confidentiality in data transmission to provide the security in a distributed system. Vieira et al. [49] have integrated knowledge and behavior analysis to detect specific intrusions. Regardless of host-based IDS, network-based IDS,

knowledge-based IDS, or behavior-based IDS, Modi et al. [25] have surveyed different intrusions affecting availability, confidentiality and integrity of cloud resources and service and have recommended IDS/IPS positioning in cloud environment to achieve desired security in the next generation networks.

Cloud computing + DOS: One of the most serious threats to cloud computing itself comes from HTTP Denial of Service or XML-Based Denial of Service attacks, Chonka et al. [2] have offered a solution to trace back through our Cloud TraceBack (CTB) to find attack source, and have introduced the use of a back propagation neutral network, which was trained to detect and filter such attack traffic.

The above research can only provide a single type of network security, they could do nothing for integrated network security service.

Cloud-based integrated network security service. provide integrated service with security protection (such as enterprise, data center, cloud computing). Existing research lays particular emphasis on middleboxes coupled with cloud computing or SDN.

Cloud computing + middleboxes: Salah et al. [38] focus on integrating the most popular types of middleboxes (e.g., IDSs, distributed denial-of-service (DDoS), FW, etc), which aims at offering an integrated set of security service for cloud computing. However, this brings a huge challenge to configurate security rules and manage so many middleboxes. CNS not only provides comprehensive security services, but also facilitates the provision of management and configuration. APLOMB, Embark [15] and Yuan et al. [51] considered that current middlebox infrastructure is expensive and complex to manage, and generates new failure modes of networks, it outsources enterprise middlebox processing to the cloud, solves security problems faced by modern enterprises. CNS as security provider on the cloud can provide APLOMB with outsourcing security services.

SDN + middleboxs: Cloudwatcher [43], which provides monitoring service for large and dynamic cloud networks, automatically detours network packets to be inspected by pre-installed network security devices. Compared to CNS, this work is lack of log and event unified management and detailed analysis of filtering rules on the middlebox, and does not solves middlebox hotspots on FDCs. CoMb [39] addresses key resource management and implementation challenges that arise in exploiting benefits of consolidation in middlebox deployments, but this work is almost difficult to achieve CoMb system due to middleboxes' closed system and incompatible architecture, and large development costs. SIMPLE [35], based on a SDN-based policy enforcement layer, takes an explicit stance to work within the constraints of legacy middleboxes and existing SDN interfaces, ensuring that the traffic is directed through the desired sequence of middleboxes and overcoming significant manual effort and operator expertise. However, this work is lack of log and event unified management and detailed analysis of filtering rules on the middlebox, and does not provide security service for cloud security.

Cloud computing + SDN + middleboxs: Split/Merge [37] can be dynamically scaled out (or in) virtual middleboxs in cloud computing, and enables load-balanced elasticity: Per-flow state may be transparently split between many replicas or merged back into one. However, this work mainly focuses on how to dynamically scaled out (or in) virtual middleboxs, and it does not provide customized network security service in cloud computing according to cloud user's security requirements and unified management.

The above cloud-based integrated network security service is lack of perfect fusion among middleboxes, cloud computing and SDN, neither provides customized network security for cloud service, nor considers maintenance costs and management complexity.

3 Design

Before the CNS design is demonstrated, it is envisioned that hardware platform, hypervisor and VMs on cloud computing are trusted and what is focused is network security of service in cloud computing. The CNS design dedicates three aspects:

- *Preventing malicious attacks from external and internal traffic*: As shown in Fig. 1b, CNS prevents network attacks from both external traffic and internal traffic to ensure network security of service domains. Whenever accessing to service domains, external traffic or internal traffic needs to pass through a desired sequence of filer domains (FDs) (e.g., FW–WAF) to prevent malicious attacks (it is also called VMs filter domains). The specific design and implementation are presented in §3.1 and §4.3.
- *Customized network security service*: Most cloud users known clearly about security requirements for the service in service domains and prefer specific measures according to their requirements. CNS adds corresponding security rules into FDs on a sequence of FDs path and forwards traffic to go through this sequence, which ensures network security. The specific design and implementation are presented in §3.2 , §4.1 and §4.3.
- *Reducing cost and complexity*: It reduces device hardware cost by migrating middleboxes to VMs in cloud computing. Furthermore, automatic analysis about cloud users' customized network security requirements and unified log management from FDs lower management complexity and costs. This section is presented in §3.3, §4.1 and §4.2.

Unlike MtoVM [7], [8] that migrates all middleboxes to the same VM, The CNS system migrates each middlebox to a separate VM. As the comparison experiment demonstrates in (§5.2), CNS gets much better performance than MtoVM. Before the design is introduced, the notation of a desired sequence of FDs is defined as filter domain chain.

Definition 1 (FDC). *Filter domain chain (FDC) represents a desired sequence of filter domains, and traffic must go through FDC to ensure their network security before arriving at servers in service domains. For example, FDC (FW → WAF) of web traffic goes through Firewall and WAF.*

3.1 Component

As shown in Fig. 1b, CNS consists of the following several components: a system domain (dom0), MDs, FDs, service domains and virtual switch (vSwitch).

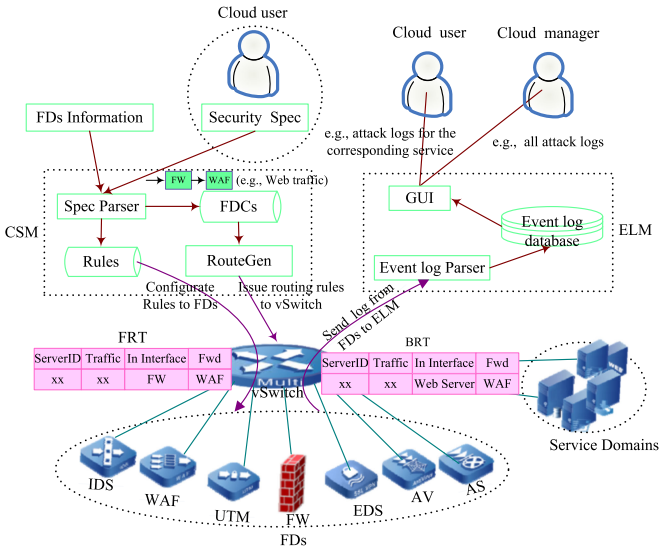


Fig. 2. CNS design.

- *Dom0* We weaken dom0 privileges, it does not have the permission to create/start and stop/destroy any domain in FDs. However, these permissions are reserved: it still has the privileges to operate every domain in service domains and management domains (MDs) and manage resources, including scheduling time-slices and I/O quotas.
- *MDs* are composed of cental security management domain (CSM) and event and log management domain (ELM). CSM has permission to create/start and stop/destroy any domain in FDs, manage and control FDs, and provide security inspection path for incoming/outgoing traffic of service domains. ELM stores and manages security events and logs from FDs and provides audit inquiry and attack statistics for cloud users and cloud administrator.
- *FDs* are a real network security inspection performer comprised of various virtual middleboxes. Network security inspection (e.g., anti-virus, filtering), decryption and encryption are realized by FDs, and this ensures that incoming/outgoing traffic to/from service domains are secure and trusted. FDs flexibly provide service domains with different security inspections according to security needs of different network-based service (called customized network security service).
- *Service Domains* hosts multiple types of service (e.g., FTP server, Web server) owned by cloud users.
- *vSwitch* receives forwarding rules from MD and forwards external and internal traffic through FDs to be filtered and inspected.

MDs and FDs cooperate jointly to provide customized network security service for cloud users, of which MDs provide incoming and outgoing traffic of service domains with their corresponding FDCs as inspection path and FDs perform security inspection when these traffic goes through FDCs. The focus of customized network security service lies in the fact that different service in service domains corresponds to different FDCs. For example, as shown in Fig. 3b, the FTP server corresponds to its FDCs, while Web server has corresponding FDCs in Fig. 3c. Due to different security requirements, the same type of service also has different FDCs. For example, the encrypted Email traffic passes through its corresponding FDCs (FW–EDS–SSL/VPN), whereas the non-encrypted Email passes through FW–EDS.

Fig. 1b shows that external and internal traffic must traverses their corresponding FDCs before arriving at service domains. When external traffic accesses the service in service domains, it is subjected to security inspection through a1, and then forwarded to service domains through a2; Internal traffic can not directly access service domains, and it must go through its corresponding FDCs (b1 and b2).

3.2 Customized Network Security Service

CNS provides customized network security service according to cloud users' various security requirements. Cloud users who know clearly about security requirements of their services in service domains only need to fill their security requirements in accordance with security spec template provided by cloud provider, and then deliver it to CNS. All the rest will be accomplished by CNS which automatically generates corresponding FDCs and security rules according to users' security spec and adds corresponding security rules into filter domains on FDCs path. The traffic must pass through FDCs to be inspected so as to ensure network security before arriving at cloud users' services.

As shown in Fig. 2, spec parser in the CSM analyzes users' spec and generates *FDCs* in both directions (incoming

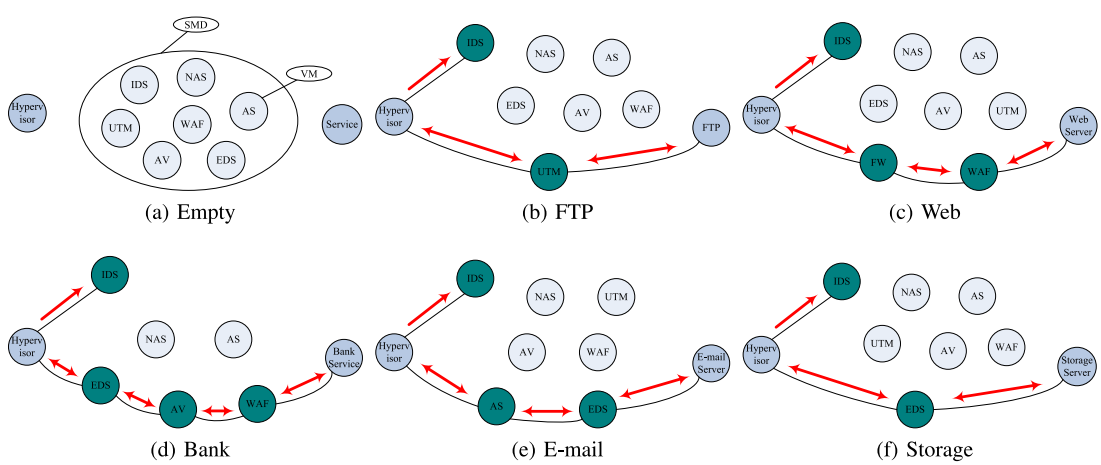


Fig. 3. Examples of customized network security service for different services.

TABLE 1
Corresponding Customized Network Security Service for Security Requirements of Different Protected Servers

Service Name	Attack	Filter Domains	Filter Domain Chains
FTP server	Session hijacking, Bounce attack, etc.	UTM	Fig. 3b
Web server	DDOS HTTP-DOS SQL injection XSS, etc	FW+WAF	Fig. 3c
Bank service	Date interception, SQL injection, Virus attack, etc	EDS+AV+WAF	Fig. 3d
E-mail server	Date interception, Spam mail, etc	EDS+AS	Fig. 3e
Storage service	Date interception, etc	EDS	Fig. 3f

and outgoing traffic) and corresponding *security rules*. These security rules are issued to FDs on FDCs path and incoming and outgoing traffic pass through these security rules on their FDCs to be filtered and inspected. For example, Web server in service domains utilizes FW and WAF to protect its network security. After analysis, security rules protecting Web server are configurated to the FW and the WAF, and FDC of its incoming traffic is FW → WAF, FDC of its outgoing traffic is WAF → FW due to the fact that most network security middelboxs are stateful and need to process both directions of a session for correctness. Refer to §5.1 for detailed content and analysis of security spec.

There are many ways to realize the function that traffic from/to service domains must go through their corresponding FDCs, a more concise way is on based on forwarding rules [23]. RouteGen in the CSM converts FDCs into forwarding rules placed in vSwtich (§5.3). In order to fast find forwarding rules, the vSwtich contains two forwarding tables: Forward Route Table (FRT) and Backward Route Table (BRT). Forwarding rules of incoming traffic is placed in the FRT, forwarding rules of outgoing traffic is placed in the BRT. The above Web server is considered as an example of forwarding: when a client accesses the web server, the vSwtich inquires forwarding rules from the FRT and Web incoming traffic is first forwarded to the FW, then the WAF, finally arrives at the web server; outgoing traffic is forwarded oppositely. In the following, a few examples of customized network security services are enumerated.

Example. It is assumed that a cloud user inquires cloud provider to provide network security of both network-layer (e.g., data link layer, network layer) and application-layer (e.g., website) for Web server in service domains. The CSM analyzes users' security requirements in conjunction with FDs topology: FW is used to protect network-layer security so as to avoid DDOS attack, UDP and ICMP flood, etc, and the WAF is used to protect application-layer security so as to avoid SQL injection, cross-site scripting attacks, etc. Therefore, an ordered combination of the FW and the WAF is adapted to protect network security of Web server required by cloud user. The specific FDCs are shown in Fig. 3c. Table 1 envisages the situation in which cloud users raise security requirements for various servers suffering from network attacks and CNS provides corresponding solutions shown in Fig. 3.

TABLE 2
Actors and Operations in the Privilege Model

Log type	Cloud Administrator	Cloud User
System logs	✓	
Audit logs	✓	
Attack logs	✓	own services ✓
Statistical reports	✓	own services ✓

Each ✓ in the table denotes that the actor can perform the corresponding operation.

3.3 Unified Management

CNS provides unified management for FDs in terms of unified configuration management and unified log management. In the traditional architecture, administrators have to face much tedious configuration management from independent vendors and different types of middleboxes. In cloud computing, if the same problem as the traditional architecture cannot be solved appropriately, it is almost an impossible task for cloud administrators to configurate and manage such a large diversity of FDs. As shown in Fig. 2, CNS can provide automated configuration and unified management to overcome these issues.

Automatic configuration CSM automatically analyzes user security spec in conjunction with the FDs topology, and then generates security rules directly configured into corresponding FDs and corresponding FDCs directly delivered to vSwitch by the method of forwarding rules, this process does not require human intervention (except for post-adjustment for special rules).

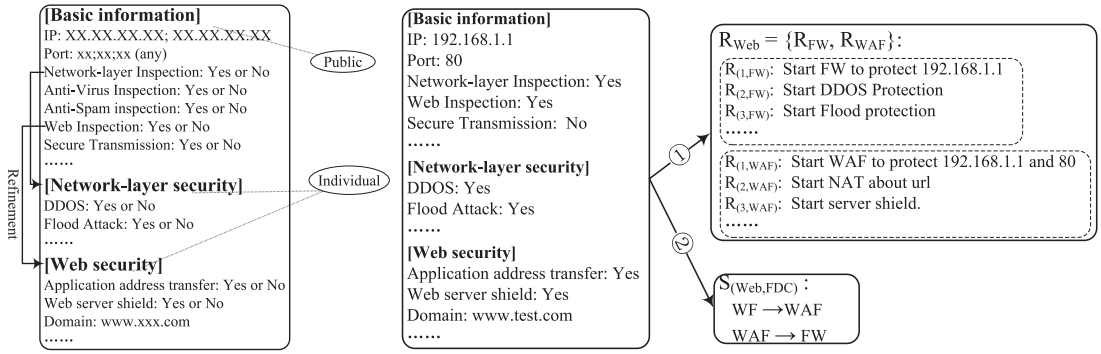
Unified log management ELM manages and counts all the logs (e.g., system logs, audit logs, attack logs) generated by FDs, and generates statistical reports based on attack logs. Logs from FDs are sent to ELM, analyzed by log analysis module in ELM, and placed in log database. To easily query logs and attack statistics, ELM provides cloud administrator and cloud users with GUI, and offers respective access privileges for different users shown in Table 2. Cloud administrator can access all the logs and statistics with high privileges, while cloud user can only access corresponding statistics and these logs are recorded when their service is under attack.

4 Implement

The above design elaborates the principle of CNS and this section presents the implementation of CNS in detail. First, CNS automatically analyzes customized security requirement spec required by cloud users, thereby generating corresponding security rules and FDCs. Second, unified logs management proves to be conducive to facilitating event and log query for cloud administrator and cloud user. Finally, the FDCs implement is presented by forwarding rules.

4.1 Customized Network Security Service Implementation

According to cloud users security requirements, CNS generates the corresponding security rules and forwarding rules to ensure services' network security. Section 3.2 shows the



(a) Customized security template (b) Security spec analysis of Web service

Fig. 4. (a) Cloud provider provides cloud users with customized security spec template in which cloud users fill in security requirements according to service security requirements. (b) CNS analyzes Web security spec from cloud users and generates filter domain chains of traffics in both direction and security rules.

principle design of customization of network security services, its focus is reflected in the implementation of security spec analysis.

Security spec template: As shown in Fig. 4a, cloud provider provides cloud users with customized security spec template, in which they fill in security requirements according to service requirements placed on service domains. The filled security spec is transmitted after encrypted in order to avoid being tampered by malicious cloud administrator [16]. The following explains some items in the template and presents some descriptive language for spec analysis. For the sake of clarity, we list some used symbols in Table 3.

IP and port: They represent protected object. IP and port fields in the basic information can be filled with one or more IP and port pairs, that is, one or more protected servers.

Algorithm 1. Spec Analysis Algorithm

```
1:  // Initialize corresponding security rules and FDC of
    each service.
2:  for each S_i ∈ S do
3:    R_i ← φ
4:    S_(i,fdc) ← φ
5:  end for
6:  R ← φ
7:  for each S_i ∈ S do
8:    // Analyze the protected Si requiring security rules
      and FDC.
9:    for each m_j ∈ M do
10:     // Add a security rule to the corresponding middelbox.
11:     while each r_k is yes do
12:       R_mj ← R_(k,mj) ∩ R_mj
13:     end while
14:     // Security rules protected by S_i.
15:     R_i ← R_mj ∩ R_i
16:     // Add the needed middlebox to FDC.
17:     if inspection item in base information is yes then
18:       S_(i,fdc) ← S_(i,fdc) ∩ m_j
19:     end if
20:   end for
21:   R ← R_i ∩ R
22: end for
```

Network-layer, Anti-Virus, Anti-Spam, Web inspection and Secure transmission: These items in the base information are important parameters to determine which virtual middleboxes to provide protection for S security requirements, and each item has a corresponding virtual middlebox. For example, R_{FW} and R_{WAF} are respectively expressed as FW security rules and WAF security rules to protect Website server S_{web}, that is, S_{web} needs security rules $R_{web} = \{R_{FW}, R_{WAF}\}$ to protect its network security.

Network-layer and Web security: They are the refinement of network-layer and Web inspection items in the basic information. Specifically, network-layer protection includes DDOS and flood attack etc. If any item in network-layer security is activated, $R_{(i,FW)}$ is used to express it, $R_{FW} = \{R_{(1,FW)}, R_{(2,FW)} \cdots R_{(n,FW)}\}$ indicates that FW consists of multiple rules $R_{(i,FW)}$. Similarly, $R_{WAF} = \{R_{(1,WAF)}, R_{(2,WAF)} \ldots R_{(n,WAF)}\}$.

CSM accepts filled and encrypted spec from a cloud user. Spec parser in CSM first decrypts the security spec, analyzes it, and then generates FDCs of traffics in both direction and security rules for the protected service domains. We show the pseudocode about the spec analysis in algorithm 1. First, security rules and FDCs of the protected objects are initialized. That is, corresponding security rules of all the protected servers are set as $R_i = \{\phi\}$ and R = $\{\phi\}$ from 2 lines to 6 lines, and corresponding FDCs of S_i is set as NULL, i.e., $S_{(i,fdc)} = \{\phi\}$. Second, security rule $R_{(k,m_j)}$ is configured to corresponding virtual middlebox m_j to protect S_i according

TABLE 3
Spec Analysis Algorithm Needs the Symbol and its Explanation

Symbol	Repression
S	The set of servers filled in security spec;
S_i	A server that is a specific IP and port pair;
R	A collection of security rules to protect S, Multiple R_i protect S by R, $R = \{R_i \mid i \in 1 \ldots n\}$;
R_i	A collection of security rules to protect S_i, and it may be dispersed in one or more virtual middleboxes on its corresponding FDCs path;
M	The set of virtual middleboxes;
m_j	Any one of M;
R_{m_j}	Security rules which m_j contains to protect S_i;
$R_{(k,m_j)}$	Security rule configured to the corresponding virtual middlebox m_j to protect S_i;
$S_{(i,fdc)}$	Corresponding FDCs of S_i;

<LogType><FDID><EventID><ServerID><SrcIP>
<SrcPort><DestIP><DestPort><Protocol><Description>

Fig. 5. Log format.

to 'yes' items in spec (lines 11-13). The same operation is performed for all the involved virtual middleboxes (lines 9-20). Third, these corresponding virtual middleboxes which provide S_i with network security are added into FDCs to protect S_i (lines 17-19). Finally, R_i is composed of security rules provided by one or more virtual middleboxes to protect S_i (lines 7-22).

Web example: Fig. 4b presents Web security spec provided by a cloud user and specific content generated by analysis. The left side in Fig. 4b shows that Web security spec enables two items in base information: network-layer and Web inspection. That is, Web server needs network-layer and Web application-layer security protection. It is obvious that a combination of FW and WAF meets security requirements of web server: First, corresponding FDCs of Web server are $S_{(Web,fdc)}$: FW → WAF and WAF → FW; Second, R_{Web} = {R_{FW}, R_{WAF}} is configured to protect S_{Web} on FDCs path. All items in network-layer security detail Web network-layer security requirements, and $R_{(2,FW)}$ and $R_{(3,FW)}$ represent specific security rules. Similarly, all the items from Web security clarify application-layer ones and specific analytical results present $R_{(2,WAF)}$, $R_{(3,WAF)}$ etc.

4.2 Log Unified Management

Log unified management is also perceived as the CNS research emphasis. If each FDs has its own management user interface (UI) as what traditional way does, it is impossible for cloud computing administrators to log in so many UIs to view attack logs and statistics information due to massive and tedious work.

Furthermore, most of the servers in service domains may require multiple FDs to protect them, inspected attack logs scattering in multiple FDs do not form integral statistics and management, therefore, it is essential for log unified management.

Algorithm 2. Log Classification Algorithm

```
1:  // Classify every log.
2:  if every log l then
3:    // l is a system log.
4:    switch (l_logtype)
5:      case system log:
6:        L_ca ← L_ca ∩ l
7:        // l log belong m_i logs.
8:        if l_.fdid = (m_i ∈ M) then
9:          L_m_i ← L_m_i ∩ l
10:       end if
11:       break
12:   // l is a attack or statistic log.
13:     case attack log and statistic log:
14:       L_ca ← L_ca ∩ l
15:       // l log belong cu_i user.
16:       if l_.serverID = (cu_i ∈ CU) then
17:         L_cu_i ← L_cu_i ∩ l
18:       end if
19:       break
20:     end switch
21: end if
```

TABLE 4
Log Classification Algorithm Needs the Symbol and its Explanation

Symbol	Repression
l	A log;
CU	All the cloud users;
L_{ca}	Log and statistics database queried by cloud administrator;
cu_i	The ith cloud user;
L_{cu_i}	Log and statistics database only queried by the ith cloud user;
L_{m_i}	Log and statistics database from the ith virtual middlebox;

For events, logs and system information from FDs, ELM performs unified management to provide cloud computing administrators with convenient management and query. In order to easily identify and standardize all logs from FDs, FDs need to abide by a unified log format shown in Fig. 5. LogType indicates log type (e.g., attack log, system log); FDID is denoted as unique FD identifier; EventID is denoted as event identifier (e.g., attack number); ServerID indicates certain domain in service domains as unique identifier to facilitate server log information statistics; SrcIP, SrcPort, DestIP, DestPort, Protocol represent quintuple flow; Description represents detailed information of the event.

After ELM receives logs, these logs are classified by log parser in order to provide access on the basis of actor permissions in Table 2. Parameters in Table 4 are introduced to facilitate the description of log classification algorithm. Algorithm 2 offers log classification. A log l arrives at ELM, If l is a system log, l is added into L_{ca} (lines 6); If l belongs to a log of m_i, l is added into the corresponding L_{m_i} (lines 8-10); If l is an attack log, l is added into L_{ca} (lines 14); if l is generated due to the attacked server owned by the ith cloud user to be attacked, l is added into L_{cu_i} (lines 16-18). After classification, cloud administrator queries all the logs from FDs, cloud users only query their owner logs generated by corresponding middleboxes when their owner servers are being attacked.

4.3 FDCs Load Balancing Implementation

We have considered load balancing of each middlebox in FDCs, and avoid each middlebox becoming a hospot.

FDCs is important part to realize customized network security service for each service in service domains. RouteGen converts FDCs and the FDs topology into forwarding rules and issues these rules to the vSwitch. CSM is considered as a SDN controller, and vSwitch is responsible for forwarding packets to/from FDs and service domains according to forwarding rules delivered by CSM. That is, network security inspection is achieved on the basis of forwarding rules in the vSwitch.

Traffic accessing to a server in service domains is divided into two types of traffic: external traffic from Internet and internal traffic between service domains in cloud computing. Incompetence internal or external traffic must go through corresponding FDCs to ensure nework security of

TABLE 5
The List of Open Source Security Softwares

Product Name	Open Source Software
FW	IPFire [11]
WAF	ModSecurity [26]
SSL/VPN	OpenSSL [32]
AS	PacketFence [44]

service domains. By default, forwarding rules from external traffic have been stored in two route tables (FRT and BRT), while there is no forwarding rules from internal traffic in the mentioned route tables. The main reason goes that there is very little communication between service domains. If forwarding rules are added into route tables, it will result in larger route table and take longer time to look up corresponding forwarding rules from the route table, which would lead to performance degradation. If the communication is established between service domains, the default route in the vSwitch forwards the first packet between them to CSM. RouteGen in the CSM generates forwarding rules by FDCs of the accessed server and issues these rules to vSwich, and subsequent traffic is forwarded in accordance with forwarding rules in the vSwitch.

During FDCs generation process, we have to consider two natural requirements: (1) Each chain FDC should have enough virtual middelboxes assigned to it, so that we retain sufficient freedom to achieve near-optimal load balancing subsequently. (2) We ensure that we have sufficient degrees of freedom; e.g., each FDC will have a guaranteed minimum number of distinct physical sequences and that no middlebox becomes a hotspot.

$$Minimize\ max\{Load_{m_j}\},\ subject\ to \tag{1}$$

$$\forall c : \sum_{c \subset Path_{FDCs}} P_{c,m_j} = 1 \tag{2}$$

$$\forall j : Load_{m_j} = \sum_{c:m_j \in Path_{FDCs}} \frac{P_{c,m_j} \times T_c \times Footprint_{c,m_j}}{ProcCap_{m_j}} \tag{3}$$

$$\forall c, j : P_{c,m_j} \in [0,1] \tag{4}$$

$$\forall j : MiddleboxUsed_{m_j} = \sum_{m_j \in Path_{FDCs}} UsedNumber_{mj} \tag{5}$$

$$\forall j : MaxMiddleboxOccurs \geq MiddleboxUsed_{m_j} \tag{6}$$

Thus, we can consider the management problem in terms of deciding the *fraction of traffic* belonging to each chain c (c $\subset$ FDCs) that each virtual middleboxes m_j has to process. Let P_{c,m_j} denote this fraction and let T_c denote the volume of traffic for each chain c. The optimization problem can be expressed by the linear program shown in Eqs. (1)–(6). Eq. (2) simply specifies a coverage constraint so that the fractional responsibilities across the virtual middleboxes on the path for each c add up to 1. Eq. (3) models the stress or load on each virtual middlebox in terms of the aggregate processing costs (i.e., product of the traffic volume and the footprints) assigned to this virtual middlebox. Here, $m_j \in Path_{FDCs}$ denotes that virtual middlebox m_j is on the routing path for the traffic in T_c. At the same time, we want to make sure that no virtual middlebox becomes a hotspot; i.e., many chains FDCs rely on a specific virtual middlebox. Thus, we model the number of chosen sequences in which a middlebox occurs and also the maximum occurrences across all middleboxes in Eqs. (5) and (6) respectively. Our objective is to minimize the value of MaxMiddleboxOccurs to avoid hotspots.

To summarize, CNS presents three important characteristics: 1) *preventing malicious attacks from external and internal traffic:* CNS prevents network attacks from external and internal traffic to ensure network security of service domains 2) *Customized network security service:* After cloud users put forwards security requirements according to their own server characteristics, CNS can be well adapted to meet security service requirements. 3) *Complexity and cost:* CNS can realize automatic configuration and management in accordance with cloud users' security spec without human intervention, which includes security rules configuration and FDCs and forwarding rules generation, and provide cloud administrators with unified logs management. Besides, CNS can provide cloud user with low-cost security service with respect to hardware and management costs of middleboxes.

5 Evaluation

In this section, there are four goals of the evaluation:

- valuate system benchmarks of CNS.
- evaluate the cost of CNS and the traditional architecture.
- evaluate maintenance and management complexity between CNS and the traditional architecture.
- evaluate performance between CNS and MtoVM, between CNS-unbind-core and CNS-bind-core and between with and without CNS.

Experimental environment Cloud platform is conducted on a Dell Server with 8 core, 3.42 GHz Intel CPU, 16GB memory. IXIA [31] and `iperf` are considered as a performance test instruments. The XEN hypervisor version is 3.4.2, and the dom0 system is fedora 16 with kernel version 2.6.31. We used a 64bit fedora Linux with kernel version 2.6.27 as our guest OS, and the vSwitch bandwidth is 1 Gigabit Ethernet; CNS uses open source security softwares shown in Table 5. For the next step, four simulation environments are installed.

- *MtoVM simulation environment:* Four kinds of open softwares in Table 5 are moved to the same VM.
- *CNS-unbind-core simulation environment:* Each software is moved to a separate VM in FDs.
- *CNS-bind-core simulation environment:* Each software is moved to a separate VM in FDs, and each VM is bound to a core, namely, each virtual middlebox runs on a separate core.
- Without security protection.

5.1 System Benchmarks

We focus on four key metrics here: the time to analyze spec, the time to install filter rules, the time to install forwarding rules, the total communication overhead at the

TABLE 6
Time and Control Traffic Overhead to Install Customized Network Security Service

Middlebox Number of each FDC	Time to Analyze Spec(ms)	Time to Install Filter Rules(ms)	Time to Install Forwarding Rules(ms)	Overhead (KB)
1	3.1	5.1	1.1	6
2	3.2	6.3	1.2	10
3	3.2	7.6	1.2	14
4	3.3	8.5	1.3	22
5	3.5	10.1	1.3	30
6	3.5	14.8	1.5	38
7	3.6	18.9	1.6	47
8	3.8	23.3	1.6	67
9	3.8	25.7	1.7	89
10	3.9	32.0	1.9	108

controller, and the maximum load on any middlebox or link in the network relative to the optimal solution. We begin by running the topology from Fig. 2 on the Emulab testbed. We did the comparison experiments according to the middleboxes number of FDCs, whose results shown are shown in Table 6.

Time to analyze spec. Table 6 shows the time taken by CNS to proactively analyze spce according to the middleboxes number from 1 to 10 of FDCs. The time to analyze increases from 3.1 ms to 3.9 ms as middlebox number in FDCs increases, but the increase is acceptable without large fluctuations. The main causes here is that the controller spends more time to analyzes more items in spec as middleboxes number of FDCs increases.

Time to install filter rules. Table 6 shows the time taken by CNS to install the filter rules for the FDCs. The time to install changes from 5.1 ms to 32.0 ms as the middlebox number of FDCs. The main bottleneck here is the controller spends mort time to install more filter rules and send more filter rules to each middlebox in FDC. We can reduce this to 70 percent overall with multiple parallel sending filter rules to middlebox.

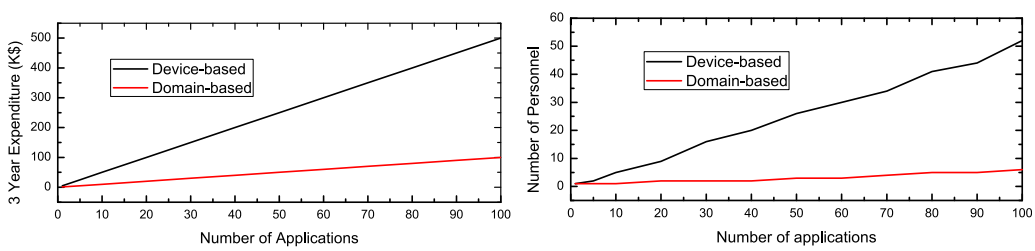
Time to install forwarding rules. The time to install forwarding rules is very short and almost does not change as as the middlebox number of FDCs. The main causes here is that it takes short time for the controller to analyze forwarding rules and sends forwarding rules only to vSwtich.

Controllers communication overhead. The table also shows the controllers communication overhead in terms of Kilobytes of control traffic to/from the controller to install filter and forwarding rules. Note that there is no other control traffic during normal operation. These numbers are consistent with the total number of rules that we need to install.

5.2 Cost and Complexity

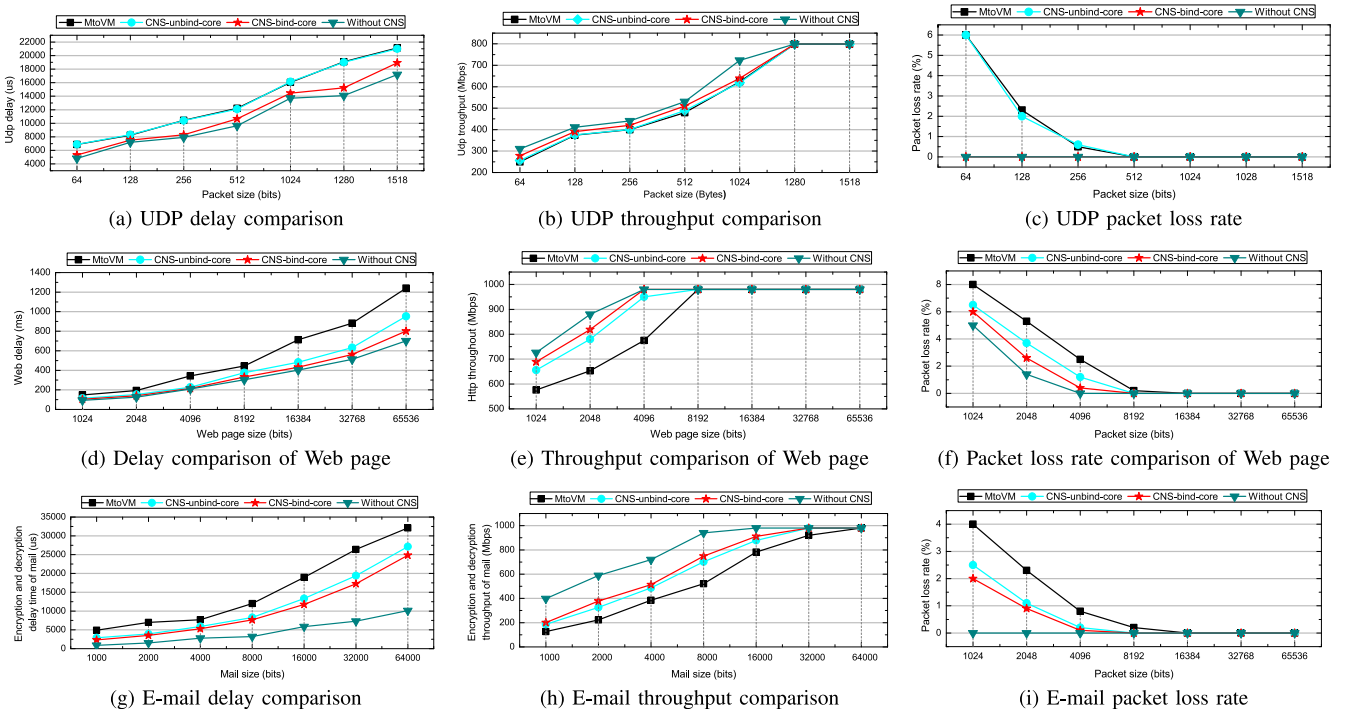
Cost. Since middleboxes (labeled as device-based) and FDs (labeled as domain-based) from independent vendors and different types of security devices or software have distinctive costs, only rough estimation rather than accurate assessment is conducted. Thus, thus the average cost of all the middleboxes and FDs are considered as their cost. Device-based and domain-based cost can be drawn according to benchmark cost [38], [42]. It can be seen from Fig. 6a that device-based cost is five times as that of domain-based. That is, the average cost of a middlebox is about $5,000, while the average cost of an FD is only $1,000. This saves the cost to a deep extent.

Complexity. Complexity focuses on configuration, maintenance and management. CNS provides security spec with automatic analysis without human intervention (except for strategy adjustment) so as to avoid security rules configuration complexity. This is especially useful for complex network security service which requires multiple middleboxes to meet full security protection, Complex network security service is a much difficult task which takes a lot of time, taking manual configuration and interactions between rules into consideration. In view of post-maintenance and post-management, the traditional architecture (labeled as device-based) is facing the complex and tedious work. For example, APLOMB [42] has conducted a survey of 57 enterprise network administrators and it is found that managing many heterogeneous middleboxes require broad expertise and consequently a large management team. Even small networks with only tens of middleboxes typically require a management team of 6-25 personnel. Unlike the traditional architecture, CNS proposes and implements automatic generation of security rules and FDCs and forwarding rules, and offers unified logs management and query. Fig. 6b presents the comparative data of management personnel between CNS and the traditional architecture in the light of the complexity: For CNS, only a few personnel are required to maintain and manage FDs, while the traditional architecture needs a management and maintenance team with large-scale personnel who rapidly grows as the number of applications increases. Especially, when the number of middleboxes reaches 100, the traditional architecture requires 50 personnel, whereas CNS only require 4 personnel.



(a) Cost comparison (b) Complexity comparison

Fig. 6. Cost and complexity comparison and between CNS and the tradition architecture.



(a) UDP delay comparison (b) UDP throughput comparison (c) UDP packet loss rate

(d) Delay comparison of Web page (e) Throughput comparison of Web page (f) Packet loss rate comparison of Web page

(g) E-mail delay comparison (h) E-mail throughput comparison (i) E-mail packet loss rate

Fig. 7. The performance comparison results between four cases: the traditional architecture, CNS-unbind-core, CNS-bind-core and without CNS.

5.3 Performance Discussion

In this section, the CNS performance is evaluated with the following goals:

- Why is CNS employed rather than MtoVM? The two are compared to prove the conclusion, and the reasons are analyzed.
- How to improve the CNS performance? What shall be done to overcome the difficulties?
- Performance overhead with CNS is evaluated to determine whether the overhead is acceptable.

Evaluation purposes are achieved by three sets of comparative experiments.

- *The first experiment* is that NCSS-unbind-core has been compared with MtoVM in term of system performance, and a better solution from comparison results can be selected , and the reasons which affect system performance are analyzed.
- *The second experiment*, some factors affecting CNS performance are overcome, and optimization results from the comparison between NCSS-unbind-core and CNS-bind-core is viewed.
- *The third experiment*, CNS performance overhead is evaluated, and related measurements on both the case with CNS-bind-core and the case without CNS-bind-core in cloud computing are performed, and whether the overhead is within acceptable range is assessed.

In order to response the comprehensive performance, every experiment presents nine sets of comparative data from three aspects of performance: latency, throughput, and packet loss rate witch are important indicator [21] of system performance about network security. The following present the three experiments.

The Comparison between MtoVM and CNS-unbind-core This is the first experiment to compare MtoVM with CNS-unbind-core about latency, throughput and packet packet loss rate. Their performances are compared by three types of service: UDP forwarding (FW), Website (FW-WAF), and Email service (FW-SSL/VPN).

In order to obtain comprehensive and correct performance assessment, UDP packets are employed with different sizes to evaluate system performance of MtoVM and CNS-unbind-core. Figs. 7a and 7b) presents experimental results about latency and throughput. In stress measurement environment, regardless of packet size from 64bit to 1528 bit, 500 Mbit/s throughput is always kept to observe packet loss. The experimental result is shown in Fig. 7c. In short, Figs. 7a, b, and Fig. 7c indicates two points: First, latency and throughput of UDP forwarding increase with the increase of their sizes. Second, MtoVM and CNS-unbind-core present the same performance, and the main reason is that UDP packets only traverse FW on the VM instead of all the virtual middleboxes on the VM although MtoVM employs multiple virtual middleboxes from Table 5 on a VM. Because CNS-unbind-core employs each virtual middlebox on a stand-alone VM, udp packets with CNS-unbind-core just go through FW according to FDC of UDP service. Therefore, MtoVM and CNS-unbind-core demonstrate the same performance in term of udp forwarding.

Website access [20], [45] based on TCP protocol needs FW and WAF to protect its network security. The experiment method is similar to the UDP forwarding except for http traffic and packets with larger size from 1024 bit to 65,536 bit. Figs. 7d, e, and Fig. 7f shows our experimental results: First, the relationship between packet size and performance is similar to the UDP forwarding. Second, regardless of latency or throughput or packet loss rate,

TABLE 7
CNS Performance Overhead Comparing to no Protective Measures in Cloud Computing

Access method	Performance	Max (%)	Min (%)	Avg (%)
UDP packet	Latency	9.1	4.2	6.4
	Throughput	13.4	0	8.8
	Packet loss rate	0	0	0
Web page	Latency	18.9	12.4	16.3
	Throughput	4.2	0	0.7
	Packet loss rate	6	0	0.9
Encrypted mail	Latency	15.7	9.2	13.1
	Throughput	5.1	0	0.8
	Packet loss rate	2	0	0.3

CNS-unbind-core is far higher than MtoVM in terms of performance.

The main reason goes like the following: the advantage of the MtoVM is that the entire inspection and filtering of Web traffic are performed only on a single VM, and this avoids the overhead of inter-VM communication and cache invalidations which may arise as shared state is accessed by multiple cores; compared with CNS-unbind-core, the MtoVM also has its own disadvantage which cloud incur overhead due to context switches and potential contention over shared resources on a single VM, especially, filtering rules and feature matching require a lot of CPU resources. Since CNS-unbind-core employs FW and WAF respectively on a stand-alone VM. Therefore, the advantages and disadvantages of CNS-unbind-core are opposite to MtoVM. A conclusion is drawn from Figs. 7d, e, and Fig. 7f) that resource contention and context switches extend greater impact than inter-VM communication and cache invalidations for MtoVM and CNS-unbind-core. If CNS is used on multiple-core virtual platform to perform parallel inspection, it is possible to overcome resource contention (especially, CPU resource) competition, which significantly improves system performance.

The importance of CPU resources for performance impact between MtoVM and CNS-unbind-core is further confirmed by e-mail encryption and decryption requiring more CPU resources. As shown in Fig. 3e, e-mail needs AS and SSL/VPN to protect its network security. Figs. 7g, h, and Fig. 7i shows CNS-unbind-core has a better performance than MtoVM in term of latency, throughput and packet loss rate. Even in the worst case, latency of MtoVM is twice than CNS-unbind-core at 64,000 bits.

In summary, regardless of UDP forwarding, Website access, Email access, Fig. 7 has showed that CNS-unbind-core realizes far higher system performance than that of the MtoVM. The main reasons is that a large number of rules and feature matching requires a lot of CPU resources which extend a greater impact than the overhead of inter-VM communication and cache invalidations for system performance. Therefore, CNS-unbind-core rather than MtoVM is adopted, which can achieve better system performance.

The Comparison between CNS-unbind-core and CNS-bind-core Resource competition, especially CPU resources, context switches, inter-VM communication and cache invalidations are regarded as main factors that affect system performance. CNS-unbind-core takes full advantage of system resources, (especially, CPU resources) and overcomes context switches over shared resources on a single VM. Since inter-VM communication between FDs makes full use of hardware-assisted I/O virtualization techniques such as single root I/O Virtualization (SR-IOV) [4] and self-assisted devices [36], it can reduce I/O virtualization overheads and achieve good performance. Therefore, inter-VM communication overhead is considered. However, multi-core scheduling constantly switches between multiple VM to lead to corresponding cache invalidations, which causes system performance degradation. In order to overcome cache invalidations, each FDs is binded to a CPU core, thus overcoming the disadvantage of cache invalidations. Fig. 7 shows our experimental results, as can be seen from nine sets of data that CNS-bind-core reflects a more superior performance than CNS-unbind-core.

The Comparison both With And Without CNS-bind-core in Cloud Computing. This is the third experiment, there are cases with the CNS and cases without the CNS in cloud computing. Fig. 7 presents the experimental comparison results indicating that the case without CNS-bind-core are more efficient than ones with CNS-bind-core. Although the case without employing CNS-bind-core to protect network security achives higher efficiency than one with CNS, it may lead to incalculable losses if no protective measures are taken to protect cloud computing security. Therefore, it is essential to protect network security of cloud computing so as to defend various attacks from the network. Even if CNS-bind-core is selected to protect cloud computing security, it is necessary to consider whether its performance overhead can be accepted. The following three experiments are still used to evaluate the performance impact with CNS-bind-core from three aspects: UDP forwarding, Website and Email service.

For UDP forwarding, Figs. 7a, b, and Fig. 7c shows NetSecCC gives little impact on system performance (specific performance overhead is shown in Table 7), compared with the case without NetSecCC, NetSecCC imposes 6.4 percent of average latency overhead (ranging from 4.4 to 9.1 percent) and 8.8 percent of average throughput drop (ranging from 0 to 13.4 percent). Packet loss rate suffers from the impact of security inspection and filtering. It is inevitable for these performance overhead to inspect and filter UDP traffic. Since UDP traffic must go through FW to be inspected and filtered before being forwarded to the UDP server in service domains. During the process, traffic is required to match hundreds of filtering rules in FW. This will take some time and result in increased latency and decreased throughput. Compared wtih MtoVM and CNS-unbind-core, CNS-bind-core has made tremendous progress.

For Website access, The results of this experiment showed in Figs. 7d, e, and Fig. 7f present CNS-bind-core has related impact on system performance. Compared with the case without CNS, latency is more affected, while throughput is hardly affected. Table 7 further illustrates that 16.3 percent of average latency overhead (ranging from 12.4 to 18.9 percent), 0.7 percent of average throughput drop (ranging from 0 to 4.2 percent), 0.9 percent of average overhead of packet loss rate (ranging from 0 to 6 percent). The main reason is like this: Web traffic must go through FW and WAF to be inspected and filtered before being forwarded to the Website

server in service domains. In this process, traffic is required to match hundreds of filtering rules in FW and thousands of signatures in WAF, which will take some time and hence result in the increased latency and the decreased throughput. In the case without CNS, Web traffic directly accesses to the Website server to avoid inspection in terms of system overhead. Therefore, compared with the cases without CNS, latency becomes longer with CNS, throughput suffers from the impact of latency. However, overall system performance with CNS is within the acceptable range.

For e-mail access, the results of this experiment showed Figs. 7g, h, and Fig. 7i) present encrypted emails with CNS are affected. The impact of longer latency, lower throughput, and bigger packet loss with CNS is mainly caused by the reason that emails must be forwarded through AS and SSL/VPN as shown in Fig. 3e. In addition, the encrypted emails require encryption processing. This will take some time and lead to performance degradation. Compared with the case without CNS, specific data with CNS on the performance overhead are shown in Table 7: the average cost of latency is 13.1 percent (ranging from 9.2 to 15.7 percent), the average cost of throughput is 0.8 percent (ranging from 0 to 5.1 percent), and the average cost of packet loss rate is 0.3 percent (ranging from 0 to 2 percent). For security services, the preceding performance overhead is acceptable.

In summary, it is found that CNS-unbind-core is a more preferred method in terms of performance by comparing both MtoVM and CNS-unbind-core. On the basis of CNS-unbind-core, it is further optimized to produce more efficient CNS-bind-core and offer more efficient customized network security service. At the same time, by the comparison of the case with CNS-bind-core and the case without CNS-bind-core in cloud computing, it is found that CNS-bind-core can provide adequate network security protection for cloud computing without sacrificing the high price of system performance.

6 Conclusion

Main problems caused by todays cloud security are high costs and performance overhead, and management complexity, especially the lack of customized network security services. In this paper, we introduced a innovative architecture called CNS, which provides customized network security for security needs of suitable cloud services as well as the qualitative benefits with respect to low performance overhead, easy to maintenance and management, and reduction in middleboxes costs. Further, we gave a specific and detailed examples and algorithms in the process of implementation in order to leverage these benefits in practice. Next, we use CNS to offer customized network security service for big data, enterprise and outsourcing security through in-depth research.

Acknowledgments

This work is partially supported by JSPS KAKENHI Grant Number JP16K00117, JP15K15976, and KDDI Foundation.

References

[1] K. Alhamazani, R. Ranjan, P. P. Jayaraman, et al., "Cross-layer multi-cloud real-time application QoS monitoring and benchmarking as-a-service framework," *IEEE Trans. Cloud Comput.*, p. 1, 2015.

[2] A. Chonka, Y. Xiang, W. Zhou, and A. Bonti, "Cloud security defence to protect cloud computing against HTTP-DoS and XML-DoS attacks," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1097–1107, 2011.

[3] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE Trans. Dependable Sec. Comput.*, vol. 10, no. 4, pp. 198–211, Jul./Aug. 2013.

[4] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *J. Parallel Distrib. Comput.*, vol. 72, no. 11, pp. 1471–1480, 2012.

[5] P. Du and A. Nakao, "DDoS defense as a network service," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2010, pp. 894–897.

[6] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 533–546.

[7] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[8] J. He, M. Dong, K. Ota, M. Fan, and G. Wang, "NetSecCC: A scalable and fault-tolerant architecture for cloud computing security," *Peer-to-Peer Netw. Appl.*, vol. 9, no. 1, pp. 67–81, 2016.

[9] J. He, M. Dong, K. Ota, M. Fan, and G. Wang, "NSCC: Self-service network security architecture for cloud computing," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, 2014, pp. 444–449.

[10] J. He, M. Dong, K. Ota, M. Fan, and G. Wang, "PNSICC: A novel parallel network security inspection mechanism based on cloud computing," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2015, pp. 402–415.

[11] IPFire. (2017). [Online]. Available: http://www.ipfire.org/

[12] D. Joseph and I. Stoica, "Modeling middleboxes," *IEEE Netw.*, vol. 22, no. 5, pp. 20–25, Sep./Oct. 2008.

[13] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An efficient elastic distributed SDN control plane," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 133–138

[14] D. Krishnan and M. Chatterjee, "An adaptive distributed intrusion detection system for cloud computing framework," in *Recent Trends in Computer Networks and Distributed Systems Security*, Berlin, Germany: Springer, 2012, pp. 466–473.

[15] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 255–273.

[16] C. Li, A. Raghunathan, and N. K. Jha, "A trusted virtual machine in an untrusted management environment," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 472–483, Oct.–Dec. 2012.

[17] H. Li, M. Dong, X. Liao, and H. Jin, "Deduplication-based energy efficient storage system in cloud environment," *Comput. J.*, vol. 58, no. 6, pp. 1373–1383, 2014.

[18] H. Li, M. Dong, K. Ota, and M. Guo, "Pricing and repurchasing for big data processing in multi-clouds," *IEEE Trans. Emerg. Topics Comput.*, vol. 4, no. 2, pp. 266–277, Apr.–Jun. 2016.

[19] C.-H. Lin, C.-W. Tien, and H.-K. Pao, "Efficient and effective NIDS for cloud virtualization environment," in *Proc. IEEE 4th Int. Conf. Cloud Comput. Technol. Sci.*, 2012, pp. 249–254.

[20] Y. L. Liu, X. Y. Bai, G. Chen, et al., "Policy-based runtime performance evaluation and validation of web services," *Acta Electronica Sinica*, vol. 38, no. 2A, pp. 182–187, 2010.

[21] M. R. Lyu and L. K. Y. Lau, "Firewall security: Policies, testing and performance evaluation," in *Proc. 24th Annu. Int. Comput. Softw. Appl. Conf.*, 2000, pp. 116–121.

[22] Y. Ma, L. Wang, A. Y. Zomaya, D. Chen, and R. Ranjan, "Task-tree based large-scale mosaicking for massive remote sensed imageries with dynamic DAG scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2126–2137, Aug. 2014.

[23] N. McKeown, et al., "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[24] M. Menzel, R. Ranjan, L. Wang, S. U. Khan, and J. Chen, "Cloudgenius: A hybrid decision support method for automating the migration of web application clusters to public clouds," *IEEE Trans. Comput.*, vol. 64, no. 5, pp. 1336–1348, May. 2015.

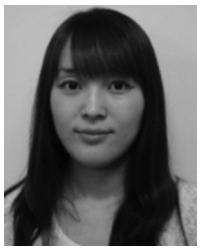
[25] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 42–57, 2013.

[26] ModSecurity. [Online]. Available: http://www.modsecurity.org/

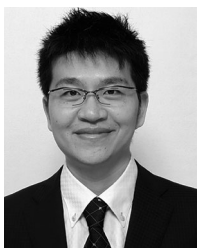
[27] A. Mohammed, S. Sama, and M. Mohammed, *Enhancing Network Security in Linux Environment*," PhD thesis, Halmstad University, Halmstad, Sweden, 2012.

[28] A. Nguyen, H. Raj, S. Rayanchu, S. Saroiu, and A. Wolman, "Delusional boot: Securing hypervisors without massive re-engineering," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 141–154.
[29] NVD. (2018). [Online]. Available: http://nvd.nist.gov/
[30] Huawei Security. (2017). [Online]. Available: http://e.huawei.com/en/products/enterprisenetworking/security
[31] IXIA. (2017). [Online]. Available: http://www.ixiacom.com/
[32] OpenSSL. (2018). [Online]. Available: http://www.openssl.org/
[33] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Comput. Sur.*, vol. 45, no. 2, 2013, Art. no. 17.
[34] McAfee SaaS Email Protection and Web Protection. (2017). [Online]. Available: http://www.mcafee.com/us/products/security-as-a-service/index.aspx
[35] Z. A. Qazi, C. C. Tu, L. Chiang, et al., "Simple-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 27–38, 2013.
[36] H. Raj and K. Schwan, "High performance and scalable I/O virtualization via self-virtualized devices," in *Proc. 16th Int. Symp. High Performance Distrib. Comput.*, 2007, pp. 179–188.
[37] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 227–240.
[38] K. Salah, J. M. Alcaraz Calero, S. Zeadally, S. Al-Mulla, and M. Alzaabi, "Using cloud computing to implement a security overlay network," *IEEE Sec. Privacy*, vol. 11, no. 1, pp. 44–53, Jan./Feb. 2013.
[39] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, Art. no. 24.
[40] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: Enabling innovation in middlebox deployment," in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, pp. 21.
[41] Imperva Cloud DDos Protection Service. (2016). [Online]. Available: http://www.imperva.com/products/wsc_cloud-ddos-protection-service.html
[42] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, 2012.
[43] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)" in *Proc. 20th IEEE Int. Conf. Netw. Protocols*, 2012, pp. 1–6.
[44] SpamAssassin. (2017). [Online]. Available: http://spamassassin.apache.org/
[45] A. Cilardo, L. Coppolino, A. Mazzeo, and L. Romano, "Performance evaluation of security services: An experimental approach," in *Proc. 5th EUROMICRO Int. Conf. Parallel, Distrib. Net.-Based Process. (PDP'07)*, Feb. 2007, pp. 387–394.
[46] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 1–11, 2011.
[47] J. Szefer and R. B. Lee, "A case for hardware protection of guest VMs from compromised hypervisors in cloud computing," in *Proc. 2nd Int. Workshop Sec. Privacy Cloud Comput.*, Jun. 2011, pp. 248–252..
[48] J. Szefer and R. B. Lee, "Architectural support for hypervisor-secure virtualization," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 437–450, Mar. 2012.
[49] K. Vieira, A. Schulter, C. B. Westphall, and C. M. Westphall, "Intrusion detection for grid and cloud computing," *Professional*, vol. 12, no. 4, pp. 38–43, 2010.
[50] H. Wu, Y. Ding, C. Winer, and L. Yao, "Network security for virtual machine in cloud computing," in *Proc. 5th Int. Conf. Comput. Sci. Convergence Inf. Technolo.*, 2010, pp. 18–21.
[51] X. Yuan, X. Wang, J. Lin, and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," in *Proc. IEEE INFOCOM 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[52] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 203–216.
[53] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Comput. Syst.*, vol. 28, no. 3, pp. 583–592, 2012.

Jin He received the BS and MS degrees in computer science and technology from the University of Electronic Science and Technology of China (UESTC), in 1998 and 2005, respectively. He is currently working toward the PhD degree in the Department of Computer Science, UESTC. He was a senior engineer with Hua Wei, China, from 2005 to 2008. He leaded two teams (unified threat management team and Web application firewall team) as the director of research and development at link-trust Co. Ltd. from 2008 to 2011. His research interests include virtualization, operating system, and cloud security.



Kaoru Ota received the BS degree in computer science and engineering from the University of Aizu, Japan, in 2006, the MS degree in computer science from Oklahoma State University, in 2008, and the PhD degree in computer science and engineering from the University of Aizu, Japan, in 2012, respectively. She is currently an assistant professor in the Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan. From March 2010 to March 2011, she was a visiting scholar with the University of Waterloo, Canada. Also she was a Japan Society of the Promotion of Science (JSPS) re-search fellow with Kato-Nishiyama Lab at Graduate School of Information Sciences at Tohoku University, Japan from April 2012 to April 2013. Her research interests include wireless networks, cloud computing, and cyber-physical systems. She has received best paper awards from ICA3PP 2014, GPC 2015, IEEE DASC 2015, and IEEE VTC 2016. She serves as an editor of the *IEEE Communications Letter*, the *Peer-to-Peer Networking and Applications* (Springer), *the Ad Hoc & Sensor Wireless Networks*, *the International Journal of Embedded Systems* (Inderscience), as well as a guest editor of the *IEEE Wireless Communications*, the *IEICE Transactions on Information and Systems*. She is currently a research scientist with A3 Foresight Program (2011-2016) funded by Japan Society for the Promotion of Sciences (JSPS), NSFC of China, and NRF of Korea. She is a member of the IEEE.



Mianxiong Dong received the BS, MS and PhD degrees in computer science and engineering from the University of Aizu, Japan. He is currently an associate professor in the Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan. Prior to joining Muroran-IT, he was a Researcher at the National Institute of Information and Communications Technology (NICT), Japan. He was a JSPS research fellow in the School of Computer Science and Engineering, The University of Aizu, Japan, and was a visiting scholar with BBCR Group, University of Water-loo, Canada supported by JSPS Excellent Young Researcher Overseas Visit Program from April 2010 to August 2011. He was selected as a foreigner research fellow (a total of three recipients all over Japan) by NEC C&C Foundation in 2011. His research interests include wireless networks, cloud computing, and cyber-physical systems. He has received best paper awards from IEEE HPCC 2008, IEEE ICESS 2008, ICA3PP 2014, GPC 2015, IEEE DASC 2015, and IEEE VTC 2016. He serves as an editor of the *IEEE Communications Surveys and Tutorials*, *the IEEE Network*, *the IEEE Wireless Communications Letters*, *the IEEE Cloud Computing*, *IEEE Access*, and *the Cyber-Physical Systems* (Taylor & Francis), as well as a leading guest editor of the *ACM Transactions on Multimedia Computing*, *the Communications and Applications*, *the IEEE Transactions on Emerging Topics in Computing* (TETC), *the IEEE Transactions on Computational Social Systems* (TCSS), *the Peer-to-Peer Networking and Applications* (Springer) and Sensors. He has been serving as the program chair of IEEE SmartCity 2015 and Symposium Chair of IEEE GLOBECOM 2016, 2017. He is currently a research scientist with A3 Foresight Program (2011-2016) funded by Japan Society for the Promotion of Sciences (JSPS), NSFC of China, and NRF of Korea. He is a member of the IEEE.

Laurence T. Yang received the BE degree in computer dcience and technology from Tsinghua University, China, and the PhD degree in computer science from the University of Victoria, Canada. He is a professor in the Department of Computer Science, St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing, big data. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation. He is a senior member of the IEEE.

Minyu Fan received the PhD degree in computer science from Xi'nan Jiao Tong University, in 1996, and was a visiting scholar with Queen University, United Kingdom, in 2005. She is a professor of computer science at UESTC. Her research interests include operating systems, distributed systems, and systems security.

Guangwei Wang is a senior software engineer, and a teacher of Computer Science at UESTC. His research interests include information security and distributed systems.

Stephen S. Yau received the BS degree in electrical engineering from National Taiwan University, and the MS and PhD degrees in electrical engineering from the University of Illinois, Urbana-Champaign. He was with the University of Florida, Gainesville, and Northwestern University, Evanston, IL. He is currently a professor of Computer Science and Engineering and the director of the Information Assurance Center with Arizona State University, Tempe. His research interests include cyber trust, cloud computing, software engineering, service-based systems, and parallel and distributed computing systems. He was the president of the IEEE Computer Society and a member of the Board of Directors of the IEEE and the Computing Research Association, and is a fellow of the American Association for the Advancement of Science. He is a fellow of the IEEE.

Doris: An Adaptive Soft Real-Time Scheduler in Virtualized Environments

Song Wu, *Member, IEEE*, Like Zhou, Xingjun Wang, Fei Chen, and Hai Jin, *Senior Member, IEEE*

Abstract—With the development of cloud computing and virtualization technologies, more and more soft real-time applications, such as Voice over Internet Protocol (VoIP) server and cloud gaming, are running in virtualized data centers. Though previous studies optimize CPU schedulers of hypervisors to support these applications in virtualized environments, there are some important challenges in designing an efficient CPU scheduler which is suitable for real-world clouds. On one hand, hypervisors do not know whether an application in a virtual machine (VM) has real-time requirements, so manually setting the scheduling parameters is a common case for CPU schedulers, which probably increases users' burden, lacks flexibility, and causes misconfigurations. On the other hand, it has been reported that most of existing CPU schedulers designed for soft real-time applications have an obvious propensity to such applications which prevents them from being applied in practical multi-tenant cloud environments. In this paper, we design and implement an adaptive soft real-time scheduler based on Xen, named *Doris*, to address these challenges. It identifies the VMs running soft real-time applications (RT-VMs) and infers their scheduling parameters according to the communication behaviors of VMs adaptively. Then, it promotes the priorities of VCPUs of the RT-VMs temporarily according to I/O events and the inferred scheduling parameters of RT-VMs to support soft real-time applications adaptively while minimizing the impacts on non-real-time applications. Finally, considering the importance of privileged entities (such as Domain0 in Xen) in I/O processing, *Doris* sets their types and scheduling parameters dynamically, which enables the adaptive scheduling of them to guarantee the performance of soft real-time applications. Our evaluation shows *Doris* can support soft real-time applications adaptively and efficiently, and only introduces very slight overhead.

Index Terms—Cloud computing, virtualization, soft real-time, CPU scheduling

1 Introduction

1.1 Motivation

With the development of cloud computing and virtualization technologies, more and more applications with different features are running in virtualized data centers, including soft real-time ones, which are the real-time applications with soft deadlines. That is to say, these applications should finish their tasks before their deadlines, but are allowed to miss a few deadlines sometimes, like streaming server [1], cloud gaming [2], VoIP server [3], and virtualized desktop systems [4].

Traditional proportional fair share CPU schedulers of hypervisors, such as Xen's Credit scheduler [5] and KVM's CFS [6], cannot support soft real-time applications well, because they usually do not consider the characteristics of these applications [7], [8], [9], [10]. Aiming at this problem, previous studies present some solutions to guarantee the performance of these applications in virtualized environments, including RT-Xen [8], laxity-based CPU scheduling [7], and parallel soft real-time scheduling [9], [10].

- *The authors are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: {wusong, zhoulike, wangxingjun, hjin}@hust.edu.cn, 350858453@qq.com.*

Manuscript received 22 Jan. 2017; revised 25 May 2017; accepted 13 June 2017. Date of publication 28 June 2017; date of current version 13 Oct. 2020. (Corresponding author: Like Zhou.)
Digital Object Identifier no. 10.1109/TSC.2017.2720732

However, these solutions cannot fit for multi-tenant cloud environments, because they need administrators to manually set scheduling parameters for VMs, which probably increases users' burden, lacks flexibility, and causes misconfigurations. Besides, they usually have an obvious propensity to soft real-time applications and ignores I/O models of hypervisors [7], [9], [10]. Actually, it is essential for clouds to support different types of applications adaptively because users may run any applications they want in VMs. In order to support soft real-time applications adaptively and efficiently in clouds, we have to face some important challenges.

First, it is a challenge for hypervisors to determine which VMs run soft real-time applications and get their scheduling parameters adaptively. First of all, in order to guarantee the performance of soft real-time applications, CPU schedulers in hypervisors need to make proper scheduling decisions according to these parameters, such as deadlines. However, it is not easy to get such parameters even in traditional native environments, because deadlines or other parameters of soft real-time applications are implicit characteristics. In order to overcome such challenge, Cucinotta et al. [11] present an approach to inferring the activation period of a soft real-time application by observing system calls generated by MPlayer, but this approach needs to modify operating systems' kernel, and the activation period could be the period of other block operations, such as disk I/O and semaphores. More importantly, virtualized systems, such as Xen [12], KVM [13], and VMware ESXi [14], add an additional layer, called hypervisor or virtual machine monitor (VMM), between guest operating systems (guest OSes) and

the underlying hardware, which causes hypervisors almost know nothing about task information and system calls in guest OSes. Although previous studies present some solutions to get task information in hypervisors [15], [16], [17], [18], they cannot get proper information for soft real-time applications because of the difficulty of getting their deadlines in virtualized environments.

The second challenge is to schedule VMs running soft real-time applications adaptively while minimizing the impacts on other applications. Clouds are multi-tenant environments, which should not allow any bias on some VMs. This is one reason for the widespread use of proportional fair shared schedulers, such as Xen's Credit scheduler. However, these schedulers ignore the characteristics of soft real-time applications, which result in inadequate performance for these applications. On the contrary, most of existing soft real-time schedulers always bias to RT-VMs [7], which may affect the performance of other applications and are not fit for practical cloud environments. Actually, soft real-time applications only need to finish their tasks before deadlines. CPU schedulers can defer the scheduling of VCPUs of RT-VMs (RT-VCPUs) to some extent and are allowed to schedule non-RT-VCPUs before RT-VCPUs. This strategy can guarantee the performance of soft real-time applications while minimizing the impacts on other applications. However, determining how long the scheduling RT-VCPUs can be deferred is a challenge.

Finally, it is a challenge for hypervisors to schedule privileged entities (such as Domain0 in Xen and QEMU [19] I/O threads in KVM) adaptively to guarantee the performance of soft real-time applications because the privileged entities play an important role in I/O processing. Privileged entities handle I/O operations for other VMs. For example, in the split driver model of Xen, all the I/O operations of VMs need to be processed by Domain0. Soft real-time applications running in cloud environments often provide services through network, such as cloud gaming, streaming server, VoIP server, and virtual desktop. If CPU schedulers only optimize for RT-VMs but leave Domain0 unchanged, all the I/O operations of RT-VMs need to be processed by a non-RT-VM (i.e., Domain0). As a result, Domain0 becomes an important factor affecting the performance of soft real-time applications. In order to support soft real-time applications efficiently, we need to schedule privileged entities adaptively. However, most of previous studies [7], [8], [9] ignore the importance of privileged entities and treat them as a non-RT-VM.

1.2 Our Contributions

In this paper, we design and implement an a**d**aptive s**o**ft **r**eal-t**i**me **s**cheduler based on Xen, named *Doris*, to support soft real-time applications that provide services through network in virtualized environments. In the following, soft real-time applications refer to the ones that provide services through network and run on virtualized environment to serve user requests. The adaptivity of *Doris* lies in three aspects. First, there are lots of applications with different characteristics running in clouds, but *Doris* can identify VMs running soft real-time applications and adaptively infer their *scheduling periods*, key scheduling parameters for *Doris*, through analyzing packets sent by VMs. Second, *Doris* can schedule these VMs adaptively according to the inferred *scheduling periods* while minimizing the impacts on non-real-time applications, which makes *Doris* suitable for multi-tenant cloud environments. Finally, *Doris* considers the importance of privileged entities and set their *scheduling periods* dynamically according to the *scheduling periods* of RT-VMs, which enables the adaptive scheduling of privileged entities and guarantees the performance of soft real-time applications further.

The main contributions of this paper are as follows.

- We propose *communication-aware period detection* approach to inferring the *scheduling periods* of RT-VMs according to their runtime characteristics. Using this approach, users can run soft real-time applications in virtualized environments without setting parameters manually, which improves the applicability of virtualization.
- We design *application-boost* mechanism to schedule RT-VMs adaptively according to the inferred *scheduling periods* of RT-VMs and I/O events while minimizing the impacts on non-real-time applications. It only promotes the priorities of RT-VCPUs temporarily and preempts the current running VCPU when needed. Otherwise, RT-VCPUs are treated as the same as non-RT-VCPUs. This mechanism is a good fit for multi-tenant cloud environments.
- We propose *privileged entity specialization* to change the types of privileged entities and set their *scheduling periods* dynamically according to the *scheduling periods* of RT-VMs. By doing so, the CPU scheduler can schedule privileged entities adaptively to guarantee the performance of soft real-time applications.
- We implement a prototype in the Xen hypervisor, named *Doris*, and use various real-world applications to evaluate its effectiveness and overhead. The experimental results show that *Doris* can support soft real-time applications adaptively and efficiently, and only introduces very slight overhead on network I/O and ignorable impacts on non-real-time applications.

2 Design

In this section, we introduce the design of *Doris* based on Xen, a popular open source hypervisor. We first describe the design principles and system overview of *Doris*. Then, we describe each component of *Doris* in detail.

2.1 Overview

Doris is an adaptive soft real-time scheduler, which can support soft real-time applications that provide services through network adaptively without setting scheduling parameters by users manually. The design of *Doris* is guided by the following principles:

- *light-weight and high precision*: the design should efficiently identify RT-VMs and infer their scheduling parameters in a light-weight and precise manner.
- *adaptivity and low impacts on other applications*: the design should schedule RT-VMs adaptively to support soft real-time applications while minimizing the impacts on other applications.
- *privileged entities support*: the design should consider how to schedule privileged entities adaptively to guarantee the performance of soft real-time applications.

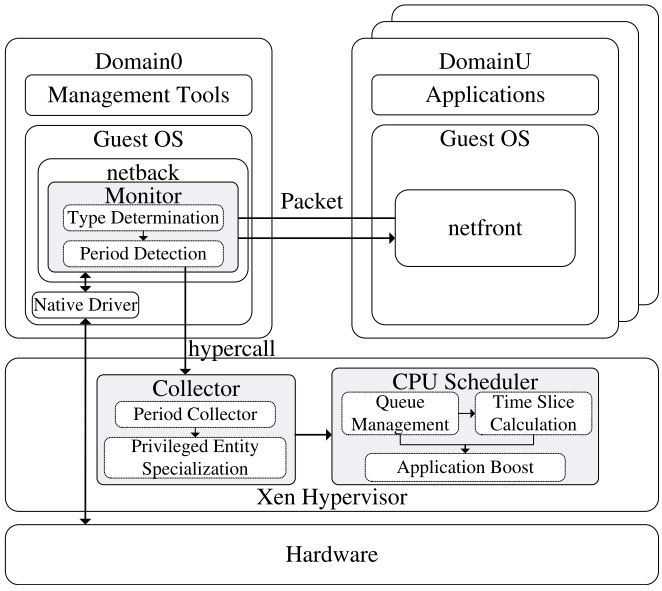


Fig. 1. Architecture of Doris, including three modules colored in gray.

We assume a VM hosts one application (may contain multiple tasks or threads), which is a common case in real-world cloud environments, and design *Doris* based on the above principles. As shown in Fig. 1, *Doris* consists of three major components: *monitor*, *collector*, and *adaptive scheduler*. Because all the packets entering and leaving DomainUs are processed by *netback* in Domain0 in Xen's split driver model, we add *monitor* to analyze all the headers of packets sent by VMs and determine which VMs are RT-VMs that sent real-time packets. (For simplicity, we call packets sent or received by soft real-time applications as real-time packets.) Then, it gets the periods of sending real-time packets by RT-VMs and uses them as the *scheduling periods* of RT-VMs because soft real-time applications running in RT-VMs often sent packets periodically. For example, a cloud gaming server needs to send video frames of games to the client periodically to guarantee that the client can see a smooth video. Actually, the *scheduling periods* of RT-VMs represent the desired scheduling parameters of soft real-time applications running in these RT-VMs under our assumption. Finally, *monitor* informs the hypervisor about the types and *scheduling periods* of VMs, which are collected by *collector*. In addition to collect such information, *collector* sets the type and *scheduling period* of Domain0 according to the collected information through *privileged entity specialization*, which enables *Doris* to schedule Domain0 adaptively to guarantee the performance of soft real-time applications. *Adaptive scheduler* schedules RT-VMs adaptively according to the collected information, which uses *application-boost* mechanism to support soft real-time applications while minimizing the impacts on other applications.

2.2 How to Identify RT-VMs and Get Their Scheduling Periods

In order to schedule VCPUs adaptively, CPU schedulers need to know the scheduling parameters of RT-VMs. In this section, we present *communication-aware period detection* approach to distinguish which VMs are RT-VMs and infer the *scheduling periods* of RT-VMs.

2.2.1 VM Type Determination

Methodology. Because CPU schedulers are critical components of hypervisors, which need to make scheduling decisions quickly, we design a lightweight approach to determine the types of VMs and infer the *scheduling periods* of RT-VMs precisely. Specifically, we devise a comprehensive method, which consists of *port-recognition* and *protocol-analysis*, to identify RT-VMs. Our proposed method uses lightweight *port-recognition* as much as possible and uses *protocol-analysis* to maintain high precision if *port-recognition* is ineffective, which can achieve our goal (i.e., lightweight and precise).

Identify RT-VMs by Port-Recognition. When soft real-time applications provide services to users, clients need to know the port number used by these applications. Some applications use well-known ports or registered ports. For example, Darwin Streaming Server (DSS) [1] uses Real Time Streaming Protocol (RTSP) to establish, control, and terminate sessions between clients and the server, which uses a well-known port 554.

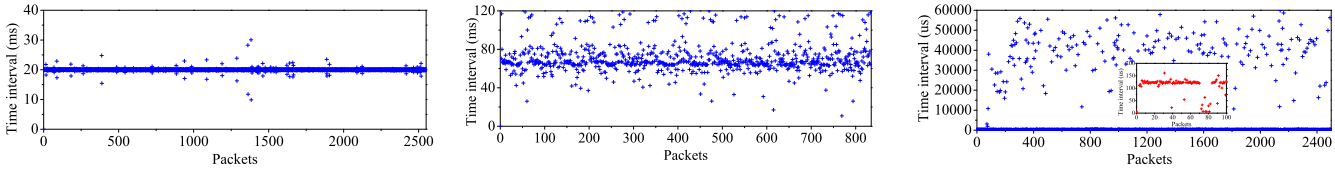
We propose *port-recognition* mechanism to analyze the header of each packet sent by VMs, and find whether its source port is related to a soft real-time application using well-known ports or registered ports. In order to find the relationship of port number and soft real-time applications quickly, we maintain a set of port number that related to soft real-time applications for each VM. For simplicity, we call this set as PORT-SET, which can be set by users manually or *protocol-analysis* mechanism.

Identify RT-VMs by Protocol-Analysis. Port-recognition mechanism has some limitations. On one hand, some soft real-time applications do not use well-known ports or change the default port because of security or other reasons. For example, VoIP servers use Real-time Transport Protocol (RTP) for media stream delivery, but the port used by RTP is not a well-known port. On the other hand, some soft real-time applications only use well-known ports to establish control connections, while establishing other connections without specific ports to transmit data. For example, DSS uses RTSP to establish control connections, but uses RTP to establish other connections to deliver video or audio streams. Therefore, we present *protocol-analysis* mechanism to analyze the header of packets, and identify real-time packets according to the characteristics of application-layer protocols.

RTSP is an application-level protocol to control streaming media servers, which is a representative explicit protocol. That is to say, the protocol provides explicit hints to identify this protocol. We can identify RTSP easily according to one of its characteristics that its header has RTSP version with a keyword "RTSP".

On the contrary, RTP is a representative implicit protocol. Though no explicit hint is available in the protocol, we can also identify RTP packets through analyzing protocol version, payload type, sequence number and synchronization source identifier of packets according to the characteristics of RTP [20].

Put Them Together. In order to identify RT-VMs in a lightweight and precise way, our proposed *comprehensive identification* mechanism preferably uses *port-recognition* mechanism to find real-time packets. When it fails, *protocol-analysis* mechanism is used to analyze the header of packets. If a packet is identified as a real-time packet, the *comprehensive identification*



(a) Asterisk (b) Darwin Streaming Server (DSS) (c) GamingAnywhere

Fig. 2. Statistics of packets sent by soft real-time applications, which shows that they send packets periodically.

mechanism updates PORT-SET with the source port of the real-time packet. Then, *port-recognition* mechanism can identify the following packets using this port. As a result, the *comprehensive identification* mechanism can reduce overhead effectively while maintaining high precision.

Note that, the types of VMs are set dynamically. When a packet sent by a VM is identified as a real-time packet, the VM is set as a RT-VM. If a RT-VM does not send any real-time packet in a time period, its type is set as non-RT-VM. Meanwhile, the port number in PORT-SET that updated by *protocol-analysis* mechanism will be cleared.

2.2.2 Period Detection

After identifying the types of VMs, we need to get the *scheduling periods* of RT-VMs. Because soft real-time applications always send packets periodically, the period of sending packets can be used as the *scheduling periods* of RT-VMs. For example, if the period of a RT-VM is 1ms, hypervisors believe soft real-time applications will send packets in every 1 millisecond. As a result, in order to guarantee the performance of these applications, CPU schedulers need to schedule VCPUs of the RT-VM according to its *scheduling period*. Note that the period of sending packets by soft real-time applications may change as time goes on and with the variation of user requests. In this paper, we present *communication-aware period detection* approach, which follows the change of sending packets' period and adjusts *scheduling periods* of corresponding RT-VMs. In this way, the underlying CPU scheduler can schedule RT-VCPUs adaptively to support soft real-time applications. In the following, we illustrate the periodicity of sending packets by soft real-time applications and describe *communication-aware period detection* approach in detail.

Periodicity of Sending Packets by Soft Real-Time Applications. We conduct experiments with Asterisk [3] (a VoIP server), DSS, and GamingAnywhere [21] (a cloud gaming server) to illustrate the periodicity of sending packets by these applications. Two machines in the same LAN are used to conduct these experiments. A machine runs these servers. The other machine is a client, which generates loads to these servers. Specifically, as to Asterisk, we use SIPp [22] to send RTP packets to it, and Asterisk echoes voices. As to DSS, we use *StreamingLoadTool* provided by DSS to simulate a user requesting a movie file for 60 seconds. We use a GamingAnywhere client to establish a connection with the GamingAnywhere server. Then, we run Wireshark [23] on the server machine to capture the packets sent by these application servers, and calculate the time interval between two adjacent packets. The test results are shown in Fig. 2.

From the test results, we can see that all these applications send packets periodically but with different periods. Asterisk needs to sample audio and send it to clients periodically to guarantee call qualities, and its period is around 20 ms as shown in Fig. 2a. DSS also needs to send packets to client periodically to guarantee that the client can see a smooth video, and its period is around 70 ms as shown in Fig. 2b. Similarly, the GamingAnywhere server also needs to send video frames of game to the client periodically, and its period is around 100 us as shown in Fig. 2c.

Communication-Aware Period Detection Approach. Monitor shown in Fig. 1 records the system time of sending real-time packets, and calculates the time interval between adjacent packets. Then it predicts the relative time of sending the next real-time packet, which is the *scheduling period* of a RT-VM. Inspired by previous studies [16], [24], we use Exponential Weighted Moving Average (EWMA), which is a light-weight predication algorithm and can tolerate transient incorrect value, to calculate *scheduling period*. As shown in Equation (1), in order to predict the current period p_t, it only needs to remember the latest historical period p_{t-1}. $diff$ is the time interval between the last two packets. α is weight with the value between 0 and 1. The more α closes to 1, the lower the weight of historical data is. When *monitor* identifies a real-time packet, it calculates p_t

$$p_t = (1-\alpha) \times p_{t-1} + \alpha \times diff. \tag{1}$$

The pseudo-code of the *communication-aware period detection* algorithm is shown in Algorithm 1. The algorithm first uses the *comprehensive identification* mechanism to analyze packets sent by VMs (line 1∼2). If a RT-VM does not send any real-time packet in a time period, the algorithm uses a hypercall added by us to set the RT-VM as a non-RT-VM. Besides, it clears the port number in PORT-SET updated by *protocol-analysis* mechanism (line 3∼6). Then, the algorithm updates PORT-SET to reduce the use of *protocol-analysis* mechanism (line 9). Finally, it sets the VM sending packets as a RT-VM if necessary, calculates p_t and updates the *scheduling period* of the RT-VM. Because frequent hypercalls introduce some overhead, the algorithm uses a timer to update the *scheduling period* periodically, which can reduce the overhead introduced by hypercalls (line 10∼14).

2.3 How to Schedule VCPUs Adaptively

In this section, we present an adaptive soft real-time scheduling algorithm to schedule VCPUs adaptively to reduce deadline misses of soft real-time applications according to the *scheduling periods* inferred by *monitor*. The key idea behind this scheduling algorithm is *application-boost* mechanism, which only promotes the priorities of RT-VCPUs temporarily and preempts the current running VCPU when needed. Otherwise, the algorithm treats RT-VCPUs and non-RT-VCPUs as the same. In the following, we first describe the Credit scheduler briefly, and *application-boost* mechanism in detail. Then, we explain the scheduling algorithm.

Algorithm 1. Communication-Aware Period Detection Algorithm

Input: packets sent by VMs
Output: types and scheduling periods of VMs.

```
port ← SOURCE_PORT(packet);
if PORT_RECOGNITION(port) && PROTOCOL_ANALYSIS(packet) then
    if vm[id] is a RT-VM && vm[id] does not send any real-time packet in a time periodthen
    set vm[id] as a non-RT-VM through a hypercall;
    clear the port number in PORT-SET updated by protocol-analysis mechanism;
    end if
    return;
end if
add port to PORT-SET;
if vm[id] is a non-RT-VMthen
    set vm[id] as a RT-VM through a hypercall;
end if
get system time and calculate p_t according to (1);
set a timer to call HYPERCALL(id, p_t) periodically;
```

2.3.1 Credit Scheduler

The Credit scheduler is a proportional fair share scheduler, which distributes CPU resources (or *credits*) to VMs at the end of each accounting period (default 30 ms) according to the *weights* of VMs specified by users.

There are three kinds of priorities in the Credit scheduler (from high to low): *boost*, *under*, and *over*. If a VCPU has remaining *credits*, its priority is *under*. Otherwise, its priority is set to *over*. When a blocked VCPU with *under* priority receives I/O events, its priority is promoted to *boost*. And then, the VCPU preempts the current running VCPU (unless they have the same priority). *Boost* priority is used to reduce the latency of I/O processing. VCPUs on a run queue are sorted according to their priorities. The scheduler picks the first VCPU from the run queue to run, and VCPUs with the same priority are scheduled in a round-robin manner. When a VCPU is descheduled, the scheduler deducts its *credits*, reassigns a priority according the remaining credits, and inserts it back to the run queue according to the new priority.

Besides, the Credit scheduler supports Symmetric Multi-Processing (SMP) platforms well. It automatically balances VCPUs across all available physical CPUs (PCPUs). When a PCPU becomes idle or its run queue has no VCPU with *boost* or *under* priority, it tries to steal a higher-priority VCPU from peer PCPU's run queue.

2.3.2 Application-Boost

In this paper, we introduce *application-boost* mechanism to guarantee the performance of soft real-time applications. It is a priority promotion mechanism for VCPUs according to the I/O behaviors and *scheduling periods* of RT-VMs. Because soft real-time applications do not need to be scheduled preferentially with permanent higher priorities and they only need to finish their tasks before deadlines, *application-boost* mechanism only promotes the priorities of VCPUs temporarily, and recovers their priorities after they are descheduled. Like *boost* mechanism in the Credit scheduler, it can minimize the impacts on non-real-time applications, which is essential for multi-tenant cloud environments.

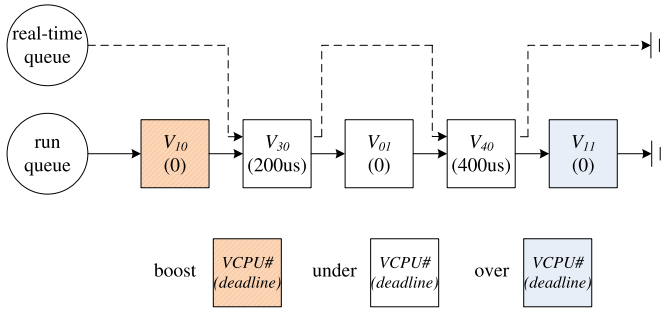


Fig. 3. Example of real-time queue, which is sorted according to the deadlines of RT-VCPUs.

In order to schedule RT-VCPUs immediately when they must be scheduled, we introduce *real-time* priority, which is the highest priority in the algorithm. VCPUs with *real-time* priorities can preempt other VCPUs with lower priorities. The priorities of RT-VCPUs can be promoted to *real-time* only if they need to be scheduled instantly, and will be degraded to lower priorities when they are descheduled. Otherwise, the treatments for RT-VCPUs are the same as non-RT-VCPUs. There are two situations that need to promote the priorities of RT-VCPUs to *real-time*, which are described as follows.

The first situation is that RT-VMs receive I/O events, such as user requests or disk operations of soft real-time applications. Typically, these applications want to process I/O events quickly. For example, when users click mouse in virtual desktop or attack a monster in cloud gaming, they hope the server can respond the requests immediately. In this case, *application-boost* mechanism is activated to shorten the response time of soft real-time applications. Compared with *boost* mechanism in the Credit scheduler, *application-boost* mechanism can also promote the priorities of VCPUs in run queues when they receive I/O events, which can achieve low response time for soft real-time applications.

The second situation is that deadlines of RT-VCPUs arrive. The deadline of a RT-VCPU is the time of being inserted into run queue plus its *scheduling period*. For example, when a RT-VCPU with a period of 1ms is inserted into run queue, its deadline is the current time plus 1 ms, that is 1 ms later. When its deadline arrives, *application-boost* mechanism is activated to schedule the VCPU. (Note that, since it is hard to estimate the worst-case execution time in multi-tenant cloud environments because of imperfect performance isolation of clouds and unpredictable user workloads running in other VMs, the deadline of a RT-VCPU in this paper is the deadline that the RT-VCPU needs to be scheduled.)

In order to activate *application-boost* mechanism when deadlines of RT-VCPUs arrive, we add a real-time queue to each PCPU, which contains all the runnable RT-VCPUs in the PCPU. The RT-VCPUs in the real-time queue are sorted by their deadlines. The first RT-VCPU of the real-time queue has the earliest deadline. When a RT-VCPU is inserted into the run queue, it is also inserted into the real-time queue according to its deadline. The behavior of inserting a RT-VCPU into a run queue is the same as inserting non-RT-VCPUs, which is inserted into the tail of the same priority subregion. Fig. 3 illustrates an example of real-time

queue. Deadlines of RT-VCPUs are positive number while non-RT-VCPUs' deadlines are 0. When a RT-VCPU with *under* priority is inserted into run queue, it is inserted behind V_{40}. If its deadline is 150us, it is inserted into the head of real-time queue. When a RT-VCPU is removed from a run queue, it is also removed from the real-time queue. If the real-time queue has RT-VCPUs, the deadline of the first VCPU in the queue, such as V_{30} in Fig. 3, is used to set a timer. When the timer times out, the deadline of the VCPU is arrived and *application-boost* mechanism is activated to schedule this VCPU.

2.3.3 Adaptive Soft Real-Time Scheduling Algorithm

The adaptive soft real-time scheduling algorithm uses *application-boost* mechanism, which relies on I/O events and *scheduling periods* detected by analyzing packets sent by VMs, to schedule VCPUs adaptively. In the following, we describe how to calculate time slice in the algorithm and explain the algorithm in detail.

In our proposed algorithm, each VCPU is allowed to run at most a time slice, but its length is not fixed. Because *application-boost* mechanism adopts timers to schedule RT-VCPUs immediately when their deadlines arrive, which introduces some overhead if timers are triggered frequently, we present *variable time slice* mechanism to reduce the use of timers. It increases the probability of scheduling the first VCPU of real-time queue just before time out of the timer. The calculation of the time slice is as follows. First, if there is no RT-VCPU in a run queue, the time slice is the default time slice, such as 30 ms in the Credit scheduler. In this case, the behaviors of the scheduler is the same as the default scheduler. Second, if the priority of the second VCPU in the run queue is *real-time*, it means at least two RT-VCPUs in a PCPU should be scheduled immediately. We set the time slice as a minimum time slice (MIN_TS), which allows the next RT-VCPU to be scheduled after at most the minimum time slice. Third, if the real-time queue is not empty, the time slice is the deadline of the first VCPU in the real-time queue minus the current system time. If the time slice is smaller than MIN_TS, it is set to MIN_TS, which can avoid too much context switches caused by too short time slice.

The pseudo-code of the algorithm is shown in Algorithm 2, which consists of two functions: *schedule()* and *timeout()*. The *schedule()* function inserts the current running VCPU into queues and picks a VCPU to run. First, if the priority of the current running VCPU is *real-time* (*RT* in the algorithm), its priority needs to be degraded before inserting it into run queue according to the principles of *application-boost* mechanism (line 3∼6). Second, if it is a RT-VCPU, the algorithm also inserts it into real-time queue (*rt-queue* in the algorithm) according to its deadline (line 7∼9), and sets a timer to call *timeout()* function if necessary (line 10∼12). Finally, the *schedule()* function picks a VCPU from the head of run queue as the next running VCPU (line 14), and calculates its time slice according to the method described in the previous paragraph (line 15∼22). The *timeout()* function is used to trigger *application-boost* mechanism. It promotes all the RT-VCPUs with the same deadline as the first RT-VCPU in real-time queue to *real-time* priorities (line 26∼29). If the priority of the first RT-VCPU is higher than the current running VCPU, the RT-VCPU preempts this VCPU through calling the *schedule()* function (line 30∼32).

Algorithm 2. Adaptive Soft Real-Time Scheduling Algorithm

Input: queue information of a PCPU
Output: scheduling decision

```
 1: function SCHEDULE()
 2:   cur ← current running VCPU;
 3:   if cur.priority == RT then
 4:     cur.priority ← UNDER or OVER;
 5:   end if
 6:   insert cur into run queue;
 7:   if cur is a RT-VCPU then
 8:     cur.deadline ← now + cur.period;
 9:     insert cur into rt-queue according to cur.deadline;
10:     if cur is the head of rt-queue then
11:       set a timer to call TIMEOUT();
12:     end if
13:   end if
14:   next ← remove a VCPU from the head of run queue;
15:   if priority of the first VCPU of run queue is RT then
16:     next.tslice ← MIN_TS;
17:   else if rt-queue is not empty then
18:     f ← the first VCPU of rt-queue;
19:     next.tslice ← MAXMIN_TS, f.deadline - now;
20:   else
21:     next.tslice ← DEFAULT_TS;
22:   end if
23:   return next
24: end function
25: function TIMEOUT()
26:   first ← remove vcpu from the head of rt-queue;
27:   remove VCPUs with the same deadline from rt-queue;
28:   set the priorities of them to RT;
29:   insert them to run queue;
30:   if first.priority > cur.priority then
31:     SCHEDULE();
32:   end if
33: end function
```

2.4 How to Treat Privileged Entities

In this section, we present *privileged entity specialization* to set the types and *scheduling periods* of privileged entities (such as Domain0 in Xen) dynamically, which makes the CPU scheduler schedule privileged entities adaptively to guarantee the performance of soft real-time applications running in RT-VMs. In the following, we take Xen as an example to describe the roles of privileged entities (i.e., Domain0) for soft real-time applications. Then, we explain *privileged entity specialization* in detail.

2.4.1 Split Driver Model of Xen

In Xen's split driver model,each *backend driver* runs in an isolated device driver (IDD) or Domain0, and each guest OS uses a *frontend driver* to communicate with the *backend driver*. All the requests and responses of the *frontend driver* are processed by the *backend driver*.

Because soft real-time applications always provide services through network, we take packet-sending operations as an example to demonstrate the important role of Domain0 for these applications. In Xen's split driver model, the packet-sending operations can be divided into two major

steps: 1) the CPU scheduler schedules the RT-VM that runs soft real-time applications sending packets through *netfront* after the scheduling of the RT-VM; 2) the CPU scheduler schedules Domain0 and *netback* in Domain0 processes the packets and sends them through the native driver.

2.4.2 Privileged Entity Specialization

As described in Section 2.4.1, Domain0 plays a very important role in the I/O processing for RT-VMs in the Xen hypervisor. However, previous studies [7], [8], [9], [10] ignore the importance of Domain0. If real-time schedulers only consider RT-VMs (i.e., DomainUs), and treat Domain0 as a non-RT-VM, I/O operations of RT-VMs (such as the responses of user requests), which may have deadlines, are handled by a non-RT-VM (i.e., Domain0) without real-time guarantee. As a result, it may cause deadline misses of soft real-time applications even if CPU schedulers schedule RT-VMs before their deadlines.

In this paper, we consider the importance of privileged entities (e.g., Domain0), and present *privileged entity specialization* to enable the adaptive scheduling of them. With *privileged entity specialization*, the types of privileged entities are set as RT-VMs if there are RT-VMs in the system. If all the VMs are non-RT-VMs, privileged entities become non-RT-VMs. However, because privileged entities have to handle the I/O operations of the RT-VMs which may have different *scheduling periods*, it is a challenge to set the *scheduling periods* of privileged entities when they are RT-VMs.

In the split driver model, I/O operations of soft real-time applications running in VMs are handled by *frontend drivers* and *backend drivers* in virtualized environments, and *backend drivers* are shared by multiple *frontend drivers* in VMs. In order to process RT-VMs' I/O operation with the earliest deadline in time, both drivers need to be scheduled before such deadline. As a result, *privileged entity specialization* sets the *scheduling period* of privileged entity as the shortest *scheduling period* of RT-VMs, which can process I/O operations with the earliest deadline in time.

3 Scheduler Implementation

We implement *Doris* based on the Credit scheduler of Xen-4.0.1. In the following, we first describe how to modify *netback* to identify RT-VMs and calculate their *scheduling periods*. Then, we add a new scheduler to Xen, called *sched_doris*, by extending the Credit scheduler.

3.1 Modification to Netback

We modify the send operations in *netback*, and add a function named *decodepackettx()* to identify whether a packet is a real-time one. In order to accelerate the port-recognition, we use a bitmap to implement PORT-SET. So the time complexity of port-recognition is O(1). Moreover, application-layer protocols can be identified in our current implementation. *Doris* can adaptively identify streaming server, VoIP server, cloud gaming, etc.

We add a hypercall in the Xen hypervisor to set the types and *scheduling periods* of VMs. If the first real-time packet sent by a VM is identified, *decodepackettx()* set the VM as a RT-VM through the hypercall. Because frequent hypercalls introduce some overhead, we implement the calculation of *scheduling period* in *netback* (the value of α is set to 0.2 empirically in Equation (1)) and tell the hypervisor in every 30 ms through the hypercall. If a RT-VM does not send any real-time packet in the last one second, the VM is marked as a non-RT-VM.

3.2 Modification to the Credit Scheduler

We implement *Doris* based on the Credit scheduler. The details of the modification are as follows.

First, a new priority named *CSCHED_PRI_TS_RT* is added as the *real-time* priority. We add structure members to record necessary information, such as *period*, *deadline*, and set the type and *scheduling period* of Domain0 dynamically according to the information of DomainUs.

Second, we implement queue operations for real-time queue. If a RT-VCPU is inserted at the head of real-time queue or the first RT-VCPU of real-time queue is removed, a timer needs to reset to call *timeout()* added by us, which triggers the preemption if necessary.

Finally, we modify *csched_schedule()*, which is responsible for selecting the next VCPU from the run queue, to realize the scheduling of RT-VCPUs before their deadlines. If the timer times out, *timeout()* promotes the priority of the first RT-VCPU of real-time queue and other VCPUs with the same deadline to *CSCHED_PRI_TS_RT*, and inserts them into run queue again. The first RT-VCPU is probably inserted at the head of run queue. As a result, *csched_schedule()* will pick the RT-VCPU as the next VCPU to run, and degrade its priority when it is descheduled. We set MIN_TS to 0.1 ms, and calculate the time slice of the next VCPU according to the descriptions in Section 2.3.3.

In summary, *Doris* does not need to modify guest OSes and inherits all the advantages of the Credit scheduler. For example, credits management in *Doris* is the same as the Credit scheduler, and *Doris* supports SMP platforms and RT-VMs with multiple VCPUs.

4 Performance Evaluation

In this section, we evaluate the performance of *Doris* by using various workloads and study the overhead introduced by *Doris*. We first describe the experimental methodology, and then present the experimental results.

4.1 Experimental Methodology

We use real-world applications to evaluate *Doris* with some user-aware metrics, which can represent deadline misses of soft real-time applications.

(1) *Experimental platform* Our evaluation is conducted on a server comprised of two quad-core 2.4 GHz Intel Xeon CPUs with Hyper-threading disabled, 24 GB memory, 1TB SCSI disk, and 1 Gbps Ethernet card. The hypervisor is Xen-4.0.1. Guest OSes are CentOS 5.5 with Linux-2.6.31.8 kernel. The configurations of VMs running on the server are 2 GB memory and 20 GB virtual disk. Unless otherwise specified, RT-VMs have 1VCPU, which run soft real-time applications. We always conduct experiments under interference configurations to verify the effectiveness of *Doris* in multi-tenant cloud environments. The interfering VMs have 8VCPUs, which run eight hungry-loop applications to consume available CPU resources. In the evaluation,

TABLE 1
Testing Applications And Benchmarks Used to Evaluate the Effectiveness and Overhead of *Doris*

Category	Application name	Description	Metric
Soft real-time applications used to evaluate the effectiveness of *Doris*.	Asterisk [3]	a classic VoIP server, which is a voice only soft real-time application.	The call quality is measured with ITU-T PESQ [25]. If it is greater than 4, it means that the VoIP service has good quality.
	DSS [1]	a widely used streaming media server, which delivers both video and voice to clients.	The stream quality is measured with average bit rate. The higher the average bit rate, the better the stream quality is.
	GamingAnywhere [21]	an open-source cloud gaming system, which is an emerging soft real-time application in cloud era and also delivers both video and voice to clients.	The quality of GamingAnywhere is measured with Frame Per Second (FPS). If the evaluated FPS is closed to the set value, the performance of GamingAnywhere is guaranteed.
Applications used to evaluate the overhead introduced by *Doris*.	ping	a tool to evaluate network latency.	The latency is measured with response time.
	netperf [26]	a tool to evaluate network throughput.	The network throughput is measured with TCP and UDP bandwidth.
	kernel compilation	a CPU-intensive application that compiles Linux kernel.	Compilation time is used to measure the speed of compilation.
	Postmark [27]	a disk I/O intensive benchmark.	The performance is measured with the processing rate of transactions.
	Apache HTTP Server	a common web server in clouds, which is network I/O intensive application.	The performance is measured with response time.

unless otherwise stated, 4 VMs are running on the server simultaneously, and one is a RT-VM while the others are interfering VMs. Machines in the same LAN act as clients to generate loads to soft real-time applications running in RT-VMs.

(2) *Scheduling approaches* We conduct experiments under several scheduling approaches as follows:
- Credit: the default CPU scheduler of Xen hypervisor.
- *Doris*: It schedules RT-VMs and Domain0 adaptively.
- *Doris w/o Dom0*: It only optimizes for RT-VMs and leaves Domain0 unchanged, which is used to demonstrate the importance of privileged entities (i.e., Domain0).
- *Doris(10us)*, *Doris(1ms)*, *Doris(10ms)*: They set their *scheduling periods* of RT-VMs and Domain0 to a fixed value (i.e., 10 us, 1 ms, and 10 ms), which are used to show the performance disadvantages of manually setting methods. 10us is a very short *scheduling period*, while the others are two representative *scheduling periods* in milliseconds.
- *Poris(1 ms)*, *Poris(10 ms)*: *Poris* is a **p**arallel **s**oft **r**eal-**t**ime **s**cheduler optimized for multi-threaded and distributed soft real-time applications presented in our previous work [10]. It also supports single-threaded ones. It uses higher priority and dynamic time slice to support these applications, and needs users to set expected latencies for VMs manually, which determines the length of time slice. However, it cannot set expected latencies below 1ms. Thus, we conduct tests under *Poris* when expected latencies are set as 1 and 10 ms.

(3) *Classification of experiments* Experiments can be divided into two categories. On one hand, experiments are conducted to evaluate whether *Doris* can support soft real-time applications adaptively. On the other hand, we conduct experiments to evaluate the overhead caused by packet analysis and the impacts on non-real-time applications. More details of testing applications are shown in Table 1.

4.2 Experiments with Soft Real-Time Applications

In this section, we evaluate the performance of *Doris* with three soft real-time applications and compare it with other related CPU schedulers.

4.2.1 Experiments with VoIP Server

We run Asterisk in a RT-VM, and SIPp [22] on the client to establish connections with the VoIP server via the SIP protocol and emulate many voice conversations using G.711 codec. We start up several concurrent calls that range from 5 to 90 to simulate the real world environment, and measure their call quality. Specifically, the concurrent calls are established at the rate of 10 calls per second instead of establishing them simultaneously. Then we can evaluate our system with dynamic server load. We conduct the test under different CPU schedulers, and the test results are shown in Fig. 4.

Seen from Fig. 4, the average PESQ is around 4.4 despite the number of concurrent calls under *Doris*. In other words,

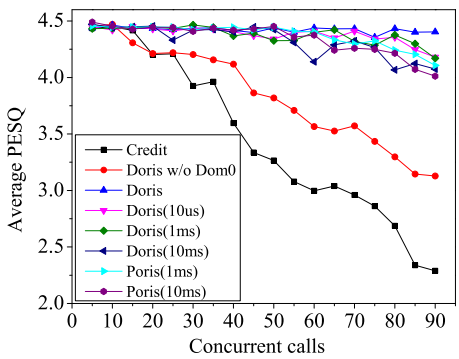


Fig. 4. The call qualities of Asterisk.

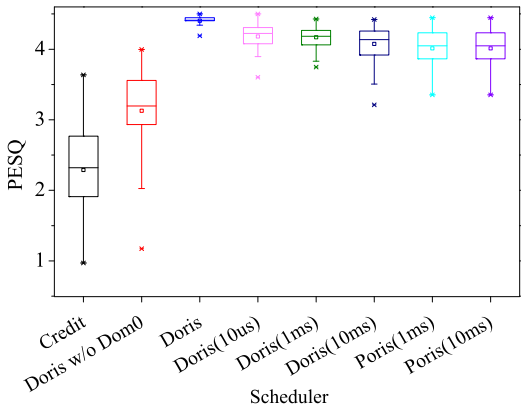


Fig. 5. The statistics of 90 concurrent calls.

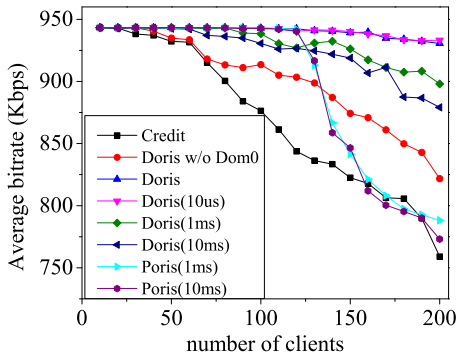


Fig. 6. Test results of DSS under different schedulers.

Doris guarantees the QoS of Asterisk without any manual settings by users. When the concurrent calls are small, all these schedulers can provide good call quality for users. However, with the increase of concurrent calls, the call qualities under the Credit scheduler and *Doris w/o Dom0* decrease sharply. Although *Doris w/o Dom0* is better than the Credit scheduler, ignoring the important role of Domain0 results in inadequate performance for Asterisk. *Doris* with manually setting periods (i.e., *Doris(10 us)*, *Doris (1 ms)*, and *Doris(10 ms)*) outperforms the Credit scheduler and *Doris w/o Dom0*. *Poris(1 ms)* and *Poris(10 ms)* show similar results. Both of *Poris* use short time slice to schedule VCPUs when we set expected latencies, which also benefits Domain0, but Domain0 cannot get the benefit of higher priority in *Poris*. *Poris* and *Doris* with manually setting periods behave very well in some cases (i.e., low concurrent calls), but are not good enough in other cases (i.e., large concurrent calls). Compared with the Credit scheduler, *Doris* improves call quality by 92.4 percent according to the average PESQ when we start up 90 concurrent calls.

We also plot the statistics of the call qualities of 90 concurrent calls under different strategies, which are shown in Fig. 5. The box in the figure shows 25th, median, and 75th percentile of statistics, and the whiskers show the minimum and maximum values of statistics. The call quality of Asterisk under *Doris* is very steady. Most of PESQ are between 4.3 and 4.5. On the contrary, no point under the Credit scheduler and *Doris w/o Dom0* is greater than 4. It also means that these schedulers cannot guarantee the quality of any call when there are lots of concurrent calls. *Poris* and *Doris* with manually setting periods behave not very well, because the qualities of some calls are not guaranteed although the average PESQ under these schedulers is greater than 4.

4.2.2 Experiments with Streaming Media Server

Streaming media server is a widely used soft real-time application. In order to guarantee the quality of video or audio services, it must transmit data to users continuously.

In this test, a RT-VM runs DSS as the streaming media server. We use *StreamingLoadTool* on the client to simulate different number of users, which range from 10 to 200. Each user requests a movie file for 60 seconds, and the bit rate of the file is 1 Mbps. The test results are shown in Fig. 6.

Seen from Fig. 6, DSS under *Doris* can provide video services with stable quality. It can adapt to the change of users. However, the quality of video services under the Credit scheduler, *Doris(1 ms)*, and *Doris(10 ms)* is decreased when the number of users increases. An interesting observation is that the performance of *Doris* is almost the same as that of *Doris(10 us)*. We believe it is mainly because DSS is an I/O intensive application too, and short time slice benefits this type of applications. Besides, we cannot simply treat 10 us as the desired *scheduling period* for DSS, because it changes according to the number of user requests, and becomes smaller with the increase of number of users. As to *Poris*, the performance of DSS is guaranteed if the number of clients is small. When the number of clients is greater than 120, its performance decreases sharply under *Poris(1 ms)* and *Poris (10 ms)*. This is probably because *Poris* ignores the importance of Domain0, which becomes a bottleneck when I/O pressure is high. As a result, the advantages of *Poris* are neutralized by the ignorance of Domain0 if soft real-time applications have lots of I/O operations. Compared with the Credit scheduler, *Doris* achieves 22.6 percent improvement according to the average bit rate when we simulate 200 clients.

4.2.3 Experiments with Cloud Gaming

Cloud gaming is an emerging gaming model, which renders an interactive gaming application remotely in the cloud and streams the scenes as a video sequence back to a thin client over the Internet [2]. In this test, we use GamingAnywhere to conduct tests.

GamingAnywhere adopts the classic client-server architecture. The GamingAnywhere server waits for incoming clients. If a GamingAnywhere client is connected with the server, the server encodes audio and video streams in real-time and sends the encoded frames to the client. The GamingAnywhere client displays real-time game screens based on the received frames, captures user inputs and sends them to the server. The server also needs to handle the user inputs in real-time. Like DSS, GamingAnywhere uses RTSP to establish connections between clients and the server, and uses RTP to deliver encoded frames if UDP is the preferred transport layer protocol.

We run a GamingAnywhere server in a RT-VM, and run OpenTTD [28], an open-source simulation game, in the RT-VM. We modify the GamingAnywhere client to display the frame rate in every second. The frame rate is set to 24 FPS in OpenTTD. We conduct the same test under different schedulers for 500 seconds, and the test results are shown in Fig. 7.

Seen from Fig. 7, the frame rate of OpenTTD under *Doris* is very steady, and the average frame rate is 23.6. Although

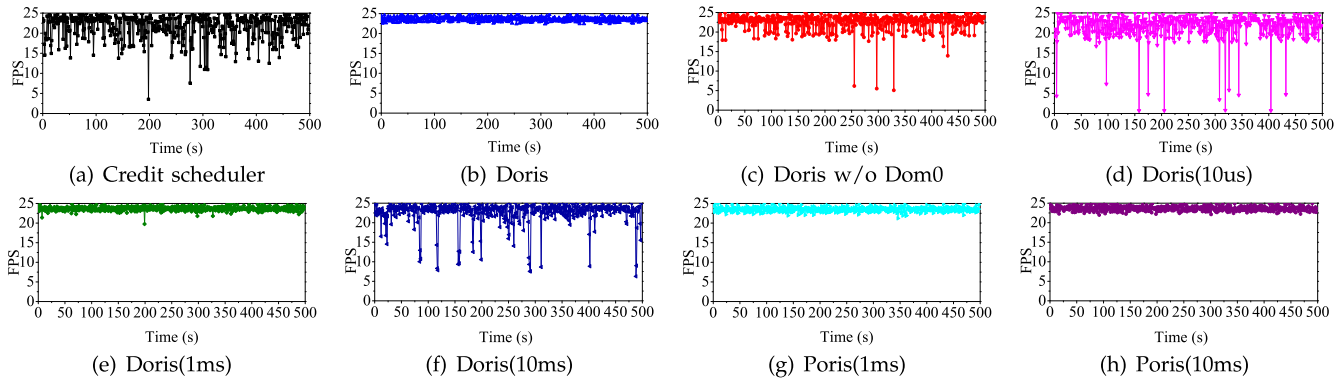


Fig. 7. Test results of GamingAnywhere under different schedulers. High and stable FPS means good user experience.

the performance under *Doris(1ms)* is closed to that of *Doris*, it still has some fluctuations. Besides, the performance is not guaranteed under the Credit scheduler, *Doris(10 us)*, and *Doris(10 ms)*. The frame rates under these schedulers are fluctuated, and many frames are lost, which affect users experience seriously. As to *Poris*, GamingAnywhere behaves well under different expected latencies. In *Poris (1 ms)*, time slice used by *Poris* is 1 ms (the minimum time slice adopted by *Poris*), while time slice is 2 ms in *Poris (10 ms)* according to the time slice calculation method described in [9]. This is why the test results are similar under these strategies. I/O operations of GamingAnywhere are relative small, so ignorance of Domain0 does not harm its performance too much.

4.2.4 Experiments with Mixed Soft Real-Time Workloads on Different VMs

In this test, we use two VMs to run soft real-time applications to verify *Doris* can adaptively support them running on different VMs. Because it is much easier to simulate multiple clients with Asterisk and DSS, we run them in these two RT-VMs and sent requests to them simultaneously from different machines. Specifically, the requests for Asterisk ranges from 5 to 90 with a step of 5 and the requests for DSS ranges from 10 to 180 with a step of 10. Since we have already evaluated the performance of Asterisk and DSS with different scheduling approaches previously, we only conduct the tests under the Credit scheduler as a baseline and *Doris* in this test, and the test results are shown in Fig. 8.

Seen from Fig. 8, the performance of Asterisk and DSS is guaranteed under *Doris* despite the number of concurrent user requests. On the contrary, the Credit scheduler performs badly with the increase of requests. This test demonstrates that *Doris* can support the scenario that multiple soft real-time applications run on different VMs and their workloads changes dynamically.

4.2.5 Experimental Results Analysis

In this section, we analyze test results of the above experiments, and conclude the following observations.

- The Credit scheduler does not consider the requirements of soft real-time applications, which causes frequent deadline misses of these applications. So it cannot support soft real-time applications well.
- *Doris* can guarantee the performance of soft real-time applications adaptively no matter how user requests change and also support multiple soft real-time applications running on different VMs. For example, as shown in Figs. 4 and 5, when the concurrent calls change, *Doris* also changes *scheduling period* accordingly, which is a prerequisite for scheduling the RT-VM in time and guaranteeing the performance of soft real-time applications. Otherwise, their performance under *Doris* will degrade when the user requests change, like *Doris* with manually setting periods.
- Domain0 plays an important role in I/O processing for DomainUs. As a result, the performance of soft real-time applications may not be guaranteed if we treat Domain0 as a non-RT-VM. The main reason is that I/O operations of RT-VMs (such as the responses of user requests), which may have deadlines, are handled by a non-RT-VM (i.e., Domain0) without real-time guaranteed.
- Specifying *scheduling periods* manually requires prior knowledge about the characteristics of soft real-time applications. The desired *scheduling periods* of various soft real-time applications are different, such as 10 us for DSS and 1 ms for GamingAnywhere under

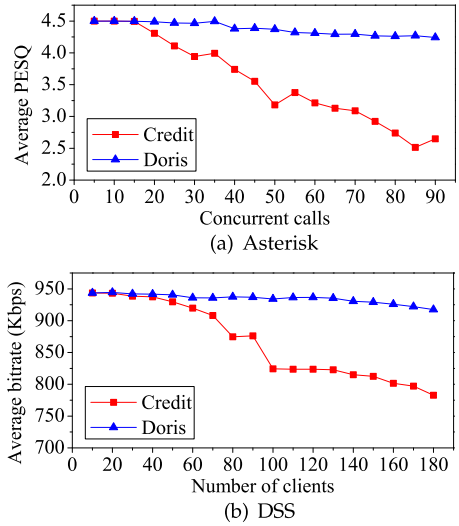


Fig. 8. Test results of Asterisk and DSS running on different VMs when clients send requests to them simultaneously.

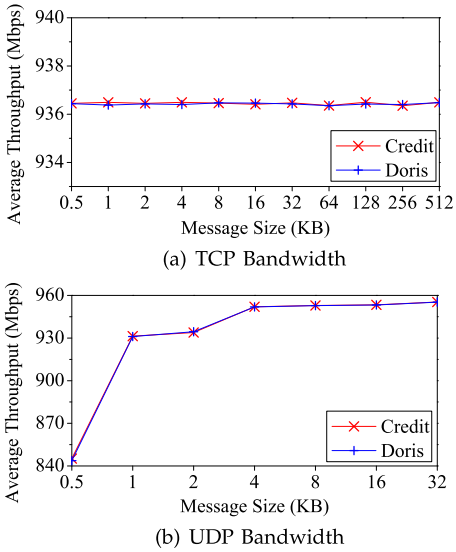


Fig. 9. Netperf throughput evaluation results.

Doris. As a result, the performance of soft real-time applications are determined by the parameters set by users. A single *scheduling period* cannot guarantee the performance of different applications, such as 1 ms in *Doris* and *Poris* as shown in Figs. 6 and 7.

- Real-time schedulers that require users to set parameters manually have different scheduling parameters, such as laxity [7], expected latency [9], period and budget [8]. Parameters with the same value (such as 10ms in *Doris* and *Poris*) have different meanings and result in different results as shown in Fig. 7f and 7h.
- Compared to schedulers with manually settings, *Doris* outperforms or has comparable performance with them. This is because *Doris* can infer the correct *scheduling periods* of soft real-time applications dynamically and the CPU scheduler schedules corresponding VCPUs adaptively according to the infer *scheduling periods*.

4.3 Impacts of Doris

Doris analyzes the headers of packets to infer the *scheduling periods* of VMs dynamically and schedules RT-VCPUs adaptively according to the *scheduling periods*. Therefore, it may introduce overhead in packets processing and impacts on non-real-time applications. In the following, we evaluate them respectively.

4.3.1 Overhead Caused by Packet Analysis

In this test, we evaluate how much overhead is caused by packet analysis. Because we only concern overhead caused by packet analysis in this test, we run a VM with one VCPU in the server and bind this VCPU to a PCPU (PCPU0), and pin all the VCPUs of Domain0 to the other PCPUs (PCPU1-7). As a result, no extra overhead is introduced by CPU scheduler, which makes test results more precise. We use ping with 0.1 second interval to measure the network latency between the VM and the client, and use netperf to evaluate both TCP and UDP network throughput. Both the tests are conducted under *Doris* and the Credit scheduler.

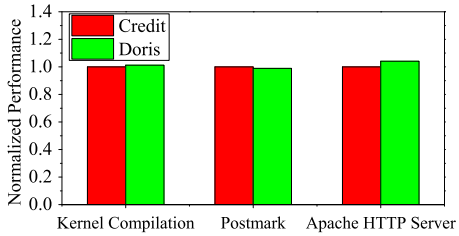


Fig. 10. The normalized performance of non-real-time applications under the Credit scheduler and Doris.

The ping test lasts for 60 seconds. The average Round-Trip Time (RTT) under the Credit scheduler and *Doris* are 0.476 and 0.481 ms respectively. As a result, *Doris* introduces very slight impacts on network latency (∼1 percent).

The netperf test also lasts for 60 seconds, and evaluates both TCP bandwidth and UDP bandwidth with different message size. As shown in Fig. 9, network bandwidth is almost the same under the Credit scheduler and *Doris*.

Although *Doris* analyzes each packet sent by DomainUs, it only introduces very slight overhead, which even can be ignored, to the network processing. This is because *Doris* analyzes packets in constant time. For example, the time complexity of *port-recognition* mechanism is O(1) because of the adoption of bitmap. The *protocol-analysis* mechanism only needs to check some fields in the packet header.

4.3.2 Impacts on Non-Real-Time Applications

In this test, we study the impacts of *Doris* on non-real-time applications, which is important for multi-tenant cloud environments.

We use 4 VMs (VM1∼VM4) to conduct this test. Each has one VCPU, and all the VCPUs are pinned to a PCPU, which can measure the impacts of *Doris* on non-real-time applications more precisely. VM1 runs Asterisk. VM2 runs non-real-time applications, including kernel compilation, Postmark, and Apache HTTP Server, which are run one by one at each test. We run hungry-loop applications on VM3 and VM4 to consume available CPU resources. We continuously send 90 concurrent calls to Asterisk, and measure the performance of non-real-time applications in VM2. The kernel compilation compiles Linux-3.3.6 source code, and the performance metric is the compilation time. Smaller compilation time is better. We set the transactions of Postmark as 400,000, and concern the processing rates of transactions. Bigger value is better. We use httperf [29] to measure the average response time of Apache HTTP Server. This test is conducted under *Doris* and the Credit scheduler. The test results are shown in Fig. 10 where the bars are normalized by the performance under the Credit scheduler.

Fig. 10 shows that *Doris* introduces very slight impacts on non-real-time applications, which also includes the overhead introduced by *Doris*. Compared with the Credit scheduler, *Doris* increases the kernel compilation time by 1.2 percent, decreases the processing rate of Postmark by 1 percent and increase the average response time of *web server* by 4.1 percent. Because *Doris* only promotes the priorities of RT-VCPUs to the highest priority temporarily when their deadlines arrive or I/O events come, non-RT-VCPUs can also run before RT-VCPUs. As a result, the impacts introduced by *Doris* on non-real-time applications are also acceptable.

5 Discussion

In this section, we discuss related issues about *Doris*.

(1) Why *Doris* uses VM-level identification?

Although a physical machine can host multiple VMs and physical resources are shared by many different types of applications, it is a common case that a VM hosts one application in cloud environments. As a result, VM-level identification is enough for *Doris* to support soft real-time applications in multi-tenant cloud environments. Besides, *Doris* supports multiple periodic tasks (belonging to a real-time application) running in a RT-VM because they share the same communication patterns. In this case, *Doris* treat the RT-VM as a black box and get an aggregation of their periods. The effectiveness is also validated in Sections 4.2.1 and 4.2.2. The VoIP server and DSS serve multiple clients, which are processed by multiple periodic tasks. As a result, compared to application-level identification, which typically requires heuristic algorithms and complicated implementations, VM-level identification is a more practical and efficient way to achieve our goal.

(2) What is the applicability of *Doris*?

As to soft real-time applications do not use well-known ports or protocols, we can implement *monitor* module depicted in Fig. 1 as a framework to calculate period, and allow developers to implement the port and protocol identification. The other option is to allow users to specify the ports in the VM configuration files, and extend *Doris* to add the ports to PORT-SET.

As to soft real-time applications that do not serve user requests, it is also possible to extend *Doris* to support them. What we need to do is to replace *monitor* module with a module that can identify periods of these soft real-time applications. The modules in the hypervisor do not need to change.

As a conclusion, although the prototype in this paper only implements the identification on some well-known ports and protocols to verify our idea, *Doris* can be extended to support more soft real-time applications.

(3) Does *Doris* violate fairness in CPU scheduling?

In Credit scheduler, fairness is guaranteed by credits management. *Doris* introduces *application-boost* mechanism to schedule VCPUs in time. Like *boost* mechanism in the Credit scheduler, *application-boost* mechanism is also built upon the original credits management, which only boost the priority of a VCPU temporarily if it has remaining credits. Therefore, the impacts on other VMs are very slight, which are validated in Section 4.3.2.

6 Related Work

This paper studies how to find the VMs running soft real-time applications and how to optimize CPU schedulers in hypervisors. In the following, we survey related work about both aspects.

Previous studies that try to get tasks or VMs information in hypervisors can be classified into two categories: intrusive and non-intrusive approach. The intrusive approach needs to modify guest OSes to obtain task information. On the contrary, the non-intrusive approach infer the statuses of guest OSes or tasks by collecting information in VMM.

Kim et al. [15] present a guest-aware priority-based VM scheduling mechanism, which takes domain's priority as the highest priority of the active tasks in it. It modifies the guest kernel to inform VMM of the priorities and statuses of its tasks. Weng et al. [18] add a monitoring module to a guest OS's kernel to monitor the waiting times of spinlocks, which is used to judge whether a task is a concurrent task. Zhao et al. [30] present *vSpec* to classify VMs into five categories by monitoring the resource-consuming characteristics, but it requires a process to collect information and is too high-level to differentiate real-time applications.

Kim et al. [17] present a method to identify I/O-bound tasks in guest OSes by observing low-level interactions between the guest kernel and hardware. They also present a method to estimate the frame rates of multimedia workloads by monitoring the frequency of frame buffer writes and a GPU command queue [16]. Hwang et al. [31] analyze port number of packets to find the VM that hosts virtual desktop infrastructure (VDI). However, it is confined to VDI and only sets the types of VMs. Chen et al. [32] present a method to infer communication-intensive applications through tracing the bus access events with the help of performance monitor unit (PMU).

In summary, previous studies cannot identify different soft real-time applications in cloud environments and infer the scheduling parameters of corresponding VMs.

As to the real-time enhancements for CPU schedulers, Lee et al. [7] introduce *laxity*, which denotes the deadline of a VM to be scheduled, to improve the performance of soft real-time applications in the Xen hypervisor. A real-time domain with low *laxity* allows its VCPUs to be inserted in the middle of the scheduler's run queue, so that the VCPUs can be scheduled within its desired deadline. RT-Xen [8], [33] introduces a hierarchical real-time scheduling framework for Xen, which bridges the gap between hierarchical real-time scheduling theory and Xen and implements several real-time scheduling algorithms. Our previous work [9], [10] presents a parallel soft real-time scheduling algorithm, which address both soft real-time constraints and synchronization problems simultaneously, to support parallel soft real-time applications in virtualized environments. However, all these studies need users to set scheduling parameters manually and ignore the importance of Domain0, which cannot fit for practical multi-tenant cloud environments.

7 Conclusion

In this paper, we designed and implemented an adaptive soft real-time scheduler, named *Doris*, in the Xen hypervisor, which can support soft real-time applications adaptively without setting types and *scheduling periods* of VMs manually. We introduced *communication-aware period detection* approach to inferring *scheduling periods* of RT-VMs and presented *application-boost* mechanism to schedule RT-VMs adaptively according to the inferred *scheduling periods* of RT-VMs and I/O events, which can minimize the impacts on non-real-time applications. Considering the importance of privileged entities (such as Domain0) in I/O processing of RT-VMs, we proposed *privileged entity specialization* to change their types and set their *scheduling periods* dynamically, which enables *Doris* to schedule privileged entities adaptively to guarantee the performance of soft real-time applications. We evaluated the effectiveness and overhead of *Doris* through various experiments. The experimental results shown that *Doris*

supports soft real-time applications well and only introduces very slight overhead. For example, compared to the Credit scheduler, *Doris* improved call quality by 92.4 percent, but only slowed down network latency by 1 percent and increased the kernel compilation time by 1.2 percent.

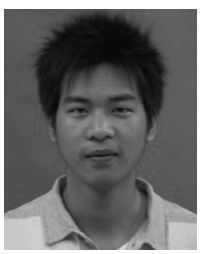
Acknowledgments

This research is supported by National Key Research and Development Program under grant 2016YFB1000501, 863 Hi-Tech Research and Development Program under grant No. 2015AA01A203, and National Science Foundation of China under grants No. 61232008 and 61472151.

References

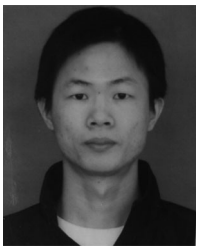
[1] Darwin streaming server. (2016). [Online]. Available: http://dss.macosforge.org/
[2] R. Shea, J. Liu, E.-H. Ngai, and Y. Cui, "Cloud gaming: Architecture and performance," *IEEE Netw.*, vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.
[3] Asterisk. (2017). [Online]. Available: http://www.asterisk.org/
[4] X. Liao, H. Li, H. Jin, H. Hou, Y. Jiang, and H. Liu, "VMStore: Distributed storage system for multiple virtual machines," *Sci. China Inf. Sci.*, vol. 54, no. 6, pp. 1104–1118, 2011.
[5] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *Performance Eval. Rev.*, vol. 35, no. 2, 2007, Art. no. 42.
[6] I. Molnar, "Linux CFS scheduler," (2007). [Online]. Available: http://kerneltrap.org/node/11737
[7] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting soft real-time tasks in the Xen hypervisor," in *Proc. 6th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, 2010, pp. 97–108.
[8] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards real-time hypervisor scheduling in Xen," in *Proc. 9th ACM Int. Conf. Embedded Softw.*, 2011, pp. 39–48.
[9] L. Zhou, S. Wu, H. Sun, H. Jin, and X. Shi, "Virtual machine scheduling for parallel soft real-time applications," in *Proc. IEEE 21st Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2013, pp. 525–534.
[10] S. Wu, L. Zhou, H. Sun, H. Jin, and X. Shi, "Poris: A scheduler for parallel soft real-time applications in virtualized environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 841–854, Mar. 2016.
[11] T. Cucinotta, F. Checconi, L. Abeni, and L. Palopoli, "Self-tuning schedulers for legacy real-time applications," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 55–68.
[12] P. Barham, et al., "Xen and the art of virtualization," in *Proc. ACM Symp. Operating Syst. Principles*, 2003, pp. 164–177.
[13] Kernel-based Virtual Machine (KVM) for Linux. (2017). [Online]. Available: http://www.linux-kvm.org
[14] V. Inc., "VMware ESX and VMware ESXi," (2017). [Online]. Available: http://www.vmware.com/products/vi/esx
[15] D. Kim, H. Kim, M. Jeon, E. Seo, and J. Lee, "Guest-aware priority-based virtual machine scheduling for highly consolidated server," in *Proc. 14th Int. Euro-Par Conf. Parallel Process.*, 2008, pp. 285–294.
[16] H. Kim, J. Jeong, J. Hwang, J. Lee, and S. Maeng, "Scheduler support for video-oriented multimedia on client-side virtualization," in *Proc. 3rd Multimedia Syst. Conf.*, 2012, pp. 65–76.
[17] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for I/O performance," in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, 2009, pp. 101–110.
[18] C. Weng, Q. Liu, L. Yu, and M. Li, "Dynamic adaptive scheduling for virtual machines," in *Proc. 20th Int. Symp. High Performance Distrib. Comput.*, 2011, pp. 239–250.
[19] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2005, pp. 41–41.
[20] RTP: A transport protocol for real-time applications. (2003). [Online]. Available: http://tools.ietf.org/html/rfc3550/
[21] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," in *Proc. 4th ACM Multimedia Syst. Conf.*, 2013, pp. 36–47.
[22] SIPp. (2014). [Online]. Available: http://sipp.sourceforge.net/
[23] Wireshark. (2017). [Online]. Available: http://http://www.wireshark.org/
[24] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu, "Optimizing virtual machine scheduling in NUMA multicore systems," in *Proc. IEEE 19th Int. Symp. High Performance Comput. Archit.*, 2013, pp. 306–317.
[25] Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. (2005). [Online]. Available: http://www.itu.int/rec/T-REC-P.862/en
[26] The Netperf Homepage. (2016). [Online]. Available: http://www.netperf.org/netperf/
[27] J. Katcher, "Postmark: A new file system benchmark," Netw. Appliance Inc., Sunnyvale, CA, USA, Tech. Rep. TR3022, 1997.
[28] OpenTTD. (2017). [Online]. Available: http://www.openttd.org/en/
[29] httperf. (2016). [Online]. Available: http://www.hpl.hp.com/research/linux/httperf/
[30] X. Zhao, J. Yin, Z. Chen, and S. He, "vSpec: Workload-adaptive operating system specialization for virtual machines in cloud computing," *Sci. China Inf. Sci*, vol. 9, no. 59, pp. 1–16, 2016.
[31] J. Hwang and T. Wood, "Adaptive dynamic priority scheduling for virtual desktop infrastructures," in *Proc. IEEE 20th Int. Workshop Quality Service*, 2012, pp. 1–9.
[32] H. Chen, H. Jin, K. Hu, and J. Huang, "Scheduling overcommitted VM: Behavior monitoring and dynamic switching-frequency scaling," *Future Generation Comput. Syst.*, vol. 29, no. 1, pp. 341–351, 2013.
[33] S. Xi et al., "Real-time multi-core virtual machine scheduling in Xen," in *Proc. Int. Conf. Embedded Softw.*, 2014, pp. 1–10.

Song Wu received the PhD degree from the Huazhong University of Science and Technology (HUST), in 2003. He is a professor of computer science and engineering with Huazhong University of Science and Technology, China. He is now served as the director of Parallel and Distributed Computing Institute, HUST. He is also served as the vice director of Service Computing Technology and System Lab (SCTS) and Cluster and Grid Computing Lab (CGCL), HUST. His current research interests include cloud computing, system virtualization, datacenter management, storage system, in-memory computing and so on. He is a member of the IEEE.



Like Zhou received the BS degree from Zhengzhou University, in 2009 and the PhD degree from HUST, in 2014. He is currently a software engineer at Google, Inc. His research interests include cloud computing, virtualization, and real-time CPU scheduling.

Xingjun Wang is currently working toward the master's degree in the Service Computing Technology and System Lab (SCTS) and Cluster and Grid Lab (CGCL), Huazhong University of Science and Technology(China). His research interests include virtualization and cloud computing.



Fei Chen received the BS degree from HUST, in 2013. He is currently working toward the PhD degree in the Service Computing Technology and System Lab (SCTS) and Cluster and Grid Lab (CGCL), Huazhong University of Science and Technology (HUST), China. His current research interests include big data, distributed system and real-time stream processing.

Hai Jin received the PhD degree in computer engineering from HUST, in 1994. He is a Cheung Kung Scholars chair professor of computer science and engineering with HUST, China. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz, Germany. He worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar with the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientists of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. He has coauthored 15 books and published more than 500 research papers. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security. He is a senior member of the IEEE and a member of the ACM.

Energy-Aware Service Selection and Adaptation in Wireless Sensor Networks with QoS Guarantee

Endong Tong, Lan Chen, and Huizi Li

Abstract—Workflow-based Service-oriented *WSN*s have recently received a lot of attention from both academia and industry. With well-defined service components, various flexible *WSN* applications can be developed. However, the key characteristic of *WSN*s is resource constraints. Sensor nodes in *WSN*s have limited storage, computation and especially limited energy. As service components in *WSN*s rely on the data gathered by sensor nodes, they are also resource-constrained. Unfortunately, traditional workflow technologies (i.e., service selection, composition and adaptation) ignore the residual energy of services. This will bring unbalanced energy consumption and furthermore shorten the network lifetime. In order to resolve this issue, we proposed an energy-aware *QoS*-guaranteed workflow management mechanism. In this mechanism, a new *QoS* model is first presented to improve the *QoS* evaluation. Then, based on this *QoS* model, an efficient service selection schema, which considers both the energy and the *QoS* of services, is proposed. Furthermore, an adaptation mechanism for balanced energy consumption is also proposed. Experimental evaluations demonstrate the capability of our proposed approach.

Index Terms—Wireless sensor network, energy-aware, quality of service, service selection, workflow management

1 Introduction

In order to support the novel paradigms of Internet of Things (*IoT*) and realize the collaboration between distributed autonomous applications in an open dynamic environment, Service-Oriented Architecture (*SOA*) [1], [2] has been used in the *IoT*'s underlying Wireless Sensor Networks (*WSN*s) [3], [4], [5], [6]. In *WSN*s, a large number of sensor devices with communication and computation capabilities will connect and interact with their surrounding environment. The functionality cells (e.g., the provisioning of online sensor data) offered by these devices can be referred to as atomic services. Hence, *SOA*-based *WSN*s logically view *WSN*s as service providers for user applications. *SOA* is a set of methodologies for designing and developing applications in the form of interoperable service components (i.e., atomic services). These atomic services are well-defined and can be reused in various applications. Among existing *SOA* technologies [7], *workflow* has become the key technology and achieved widely use due to its efficiency and practicability. A workflow [8] can be composed of a sequence of atomic services to satisfy specified input and output requirements.

In real applications, the granularity of an atomic service is usually small. For example, there will be an atomic service that providing the sensed data of one specific sensor node. *WSN*s have the great ability of data gathering, which can support various types of services. Hence, in practical *SOA*-oriented *WSN* applications, the number of atomic services is huge and there may exist many atomic services which implement the same function. Accordingly, how to achieve the optimal atomic service selection, has become a very important research issue. In actual service systems, users will not only have functional requirements (i.e., implement the required function) but also have non-functional constraints (i.e., guarantee better experience). For this issue, Quality of Service (*QoS*), which refers to the common non-functional characteristics of atomic services (e.g., execution time, cost, reliability, availability and reputation [9]), has been exploited [10]. Recently, many efforts [11], [12], [13] have been made in order to choose the best candidate atomic services that satisfy multiple *QoS* requirements.

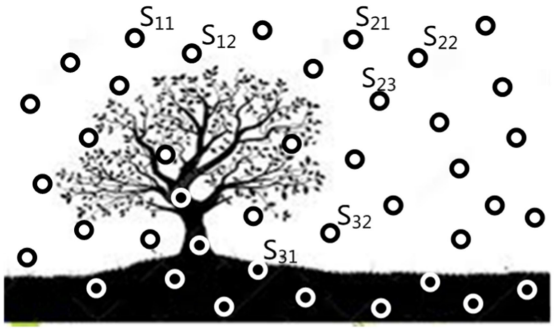
During the running process of a workflow, the *QoS* of some component atomic services may degrade. This may result in that the composed workflow is no longer *QoS* guaranteed [14], [15]. Hence, a dynamic adaptation for this workflow is necessary, which is the key issue of workflow management.

Much work has been done on workflow management in web-based applications. While work on workflow management (especially the energy efficient service selection and adaptation mechanism with *QoS* guarantee) in *WSN*s is fewer. In *WSN*s, energy is the most important resource. Unlike traditional *SOA* applications which only try to maximize the overall *QoS* utility, the main aim of *SOA*-oriented *WSN*s is to minimize the energy consumption and furthermore prolong the network lifetime.

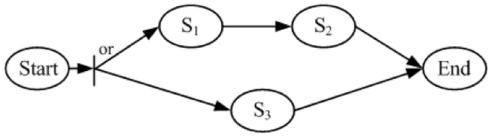
Hence, in our paper, we will extend the traditional workflow management by taking each service's residual energy

• *The authors are with the Institute of Microelectronics, Chinese Academy of Sciences, Beijing 100029, P.R. China. E-mail: {tongendong, lihuizi}@ime.ac.cn, chenlan@gmail.com.*

Manuscript received 31 Oct. 2016; revised 3 Aug. 2017; accepted 28 Aug. 2017. Date of publication 5 Sept. 2017; date of current version 13 Oct. 2020. (Corresponding author: Endong Tong.)
Digital Object Identifier no. 10.1109/TSC.2017.2749227



(a) Environmental Monitoring Application Scenario



(b) Service Composition Instance

Abstract Service	Concrete Service	Execution Time	Cost	Availability	Residual Energy
S_1	S_{11}	100	60	0.95	0.85
	S_{12}	80	52	0.93	0.41
S_2	S_{21}	130	110	0.96	0.75
	S_{22}	116	92	0.98	0.70
	S_{23}	150	117	0.95	0.75
S_3	S_{31}	195	196	0.96	0.76
	S_{32}	210	180	0.91	0.80

(c) Atomic Service QoS Attributes

Fig. 1. Example of QoS-oriented service composition.

into consideration, in order to be adapted to the resource-constrained *WSN*s.

1.1 Motivating Example

We begin with an environmental monitoring application to show why the *energy* is important for service composition. Fig. 1a shows a practical application scenario, in which many sensor nodes are deployed to monitor the surrounding environment. Fig. 1b shows an environmental monitoring service composition instance which has two possible execution pathes: $\{S_1, S_2\}$ and $\{S_3\}$. Each abstract service S_i can be replaced with one of its concrete atomic services ($\{S_{ij}\}$), which are same on function but different on *QoS*. According to a user's functional requirements, a workflow which consists of a set of abstract services will be constructed. While the selection of concrete service for each abstract service will be realized by our service computing engine. The *QoS* attribute values of each concrete atomic service are shown in Fig. 1c.

It is worth to note that, *residual energy* is usually not a concern of a general user. Therefore, a general user will not put forward requirements for *residual energy*. We assume that a user's *QoS* requirements are:

- $\texttt{execution time} \leq 220$
- $\texttt{cost} \leq 160$
- $\texttt{availability} \geq 0.90$

Due to the fact that each user has different preferences for different *QoS* attributes, a utility function is used. The utility function is defined as $Utility(cp_i) = \sum_{j=1}^{n} w_j \cdot QoS_{ij}$, in which cp_i refers to the *i*th composition plan, QoS_{ij} refers to the value of the *j*th *QoS* attribute of cp_i and w_j refers to the preference weight for QoS_{ij}. However, different *QoS* attributes may have different orders of magnitude. In order to resolve this inconsistency, a normalization process should be first done. In this example, we set $w_1 = 0.3$ for *execution time*, $w_2 = 0.7$ for *cost* and $w_3 = 0.5$ for *availability*. The optimal service selection plan, which maximizes the sum of all component services' utilities, is $\{S_{12}, S_{22}\}$ with a total utility of 117, an execution time of 196, a cost of 144 and an availability of 91 percent. However, the residual energy of atomic service S_{12} is only 41 percent. Long time execution of S_{12} will further decrease its energy and furthermore make S_{12} energy exhausted, which may affect the network connectivity. In addition, the service deployed on S_{12} will be disabled, which may bring the dysfunction of the environmental monitoring application. If we take the residual energy into consideration, the optimal service selection plan should be $\{S_{11}, S_{22}\}$ with a total utility of 109, an execution time of 216, a cost of 152 and an availability of 93 percent, while the residual energy of S_{11} is 85 percent, S_{22} is 70 percent. The new plan also fulfills the user's *QoS* requirements, but it has higher residual energy compared with the former one.

1.2 Research Issues

From the above example, we come to the conclusion that the *residual energy* of atomic service is also an important factor that should be considered for workflow construction and management in resource constrained *WSN*s. Therefore, energy-aware policies should be adopted in order to eliminate the existing variations in the energy levels in order to prolong the network lifetime. In this paper, we aim to realize the energy-aware *QoS*-guaranteed workflow management in *WSN*s by addressing the following key issues:

- Energy efficient service selection with *QoS* guarantee. Based on the consideration of *QoS* attributes and atomic services' state (e.g., running status and energy), we will develop a constrained multi-objective service selection mechanism, which can satisfy users' *QoS* requirements and at the same time own high energy.
- Multi-user lifetime-oriented dynamic workflow management. During the workflow running process, the residual energy of an atomic service will change dynamically, and there may also be new users' requests arrive. Hence, we will monitor the workflow state at runtime and re-plan the workflow plan in case of necessity, in order to prolong the network lifetime.

The rest of the paper is organized as follows. Section 2 discusses the preliminaries and related work. Then, Section 3 proposes the energy-aware atomic service selection approach, followed by *QoS* degradation and reusing based workflow adaptation mechanism in Section 4. Section 5 provides experiments that illustrate the benefits of the proposed workflow management scheme. Finally, Section 6 concludes this paper and discusses future research.

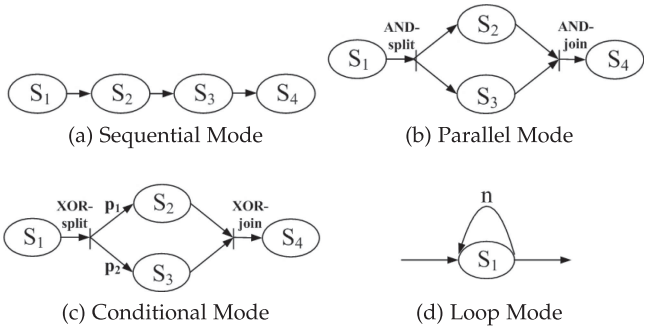


Fig. 2. Four Basic Composition Modes

2 Preliminaries and Related Work

2.1 Preliminaries

2.1.1 QoS Model

Before the service selection, an appropriate *QoS* model is necessary. Alrifai et al. [11] mentioned that QoS can be categorized into two types: *general QoS* and *domain-specific QoS*. *General QoS* means the *QoS* resolved in common applications, such as response time, cost and availability; *Domain-specific QoS* means the *QoS* resolved in special application domains, such as "data accuracy", "accessibility" and "timeliness" in *WSN* applications. However, there also exists another type of *QoS* that users may not concern, but it plays an important role on the application experience. This type of *QoS* can be categorized into *Potential QoS*. Accordingly, for resource constrained *WSN*s, we propose an extended *QoS* model which consists of *general QoS*, *domain-specific QoS* and *potential QoS*. In this paper, the *potential QoS* mainly refers to *energy*, which consists of *residual energy* and *energy consumption rate*.

In *WSN*s, each atomic service will rely on the data gathered by some sensor nodes. Although the *energy* is the attribute of sensor nodes, it can affect the state of atomic services. Hence, the *energy* is also the attribute of atomic services. For simplicity, we use the *service energy* (denoted as E_S, the residual energy of atomic service S and ΔE_S, the energy consumption rate of atomic service) in the following part of this paper. To be pointed out that, an atomic service S may rely on either one or more than one type of sensor nodes. Then, if S relies on one type of sensor nodes which has n redundant sensor nodes, then

$$(\Delta)E_S = max\{(\Delta)e_1, (\Delta)e_1, \ldots, (\Delta)e_n\} \tag{1}$$

As energy consumption means the decrease of energy, ΔE_S is a negative value. While if S relies on m types of sensor nodes and each type owns n redundant sensor nodes, then

$$\begin{aligned}(\Delta)E_S = min\{&max_1\{(\Delta)e_1, (\Delta)e_1, \ldots, (\Delta)e_n\},\\ &max_2\{(\Delta)e_1, (\Delta)e_1, \ldots, (\Delta)e_n\}, \ldots,\\ &max_m\{(\Delta)e_1, (\Delta)e_1, \ldots, (\Delta)e_n\}\}\end{aligned} \tag{2}$$

where e_i refers to the residual energy of the ith sensor node and Δe_i refers to the energy consumption rate of the ith sensor node. In practical *SOA*-oriented *WSN*s, there are various methods that atomic services can be integrated to build an workflow. The four basic modes are (1) sequential; (2) parallel; (3) conditional; and (4) loop, as shown in Fig. 2.

Among the four modes, the sequential mode is the most fundamental one. In this paper, all other modes will be converted to the sequential mode. For a workflow, the performance is evaluated in terms of its end-to-end quality. Hence, we will discuss the corresponding computing methods for the four basic composition modes. Many research has provided the computing methods for different *QoS* attributes according to the specific mode, while the computation of *energy* has been ignored. The energy of a workflow is determined by the energy of each individual atomic service as well as its mode. For the four basic modes in Fig. 2, the energy of the composite workflow can be computed as following:

$$(\Delta)E_W = \begin{cases} min((\Delta)E_{S_1}, (\Delta)E_{S_2}, (\Delta)E_{S_3}, (\Delta)E_{S_4}), \\ \qquad for\ sequential\ and\ parallel\ modes; \\ min((\Delta)E_{S_1}, (p_1 \cdot (\Delta)E_{S_2} + p_2 \cdot (\Delta)E_{S_3}), (\Delta)E_{S_4}), \\ \qquad\qquad for\ conditional\ mode; \\ (\Delta)E_{S_1},\ \ for\ loop\ mode; \end{cases} \tag{3}$$

Here, p_1 and p_2 indicate the probability that a user will choose service S_2 and S_3 respectively.

According to the characteristics of different *QoS* attributes, the General *QoS* and Domain-specific *QoS* can be divided into *Multiplicative QoS* and *Additive QoS*. The corresponding aggregation approaches are listed in Table 1.

2.1.2 Network Model

We will first give an overview of the network model for better understanding our study scenario. In our paper, the mentioned *WSN*s are the networks of wireless smart sensor nodes in the *IoT*, which have the following characteristics: (1)The transmitted sensor data are usually short messages, such as a temperature value or a location message; (2)The transmission of sensor data is usually event-driven. Hence, there is no need to intercurrently maintain large numbers of connections between sensor nodes and their corresponding sink node. Then, the sensor networks in our paper use hierarchical sink nodes. Sink nodes are more powerful devices, often a personal computer, that are in charge of gathering the collected sensing data, further processing them, and make them available to *WSN* applications. As shown in Fig. 3, each sensor node will transmit its sensed data to its cluster head directly, and this sink node then forwards the received data to its parent sink node, then finally to the root sink node.

TABLE 1
QoS Aggregation Approaches

QoS Type	Sequential Mode	Parallel Mode	Conditional Mode	Loop Mode
Multiplicative QoS	$\prod_{\{S_1 \sim S_4\}} QoS_{S_i}$	$\prod_{\{S_1 \sim S_4\}} QoS_{S_i}$	$(p_1 \cdot QoS_{S_2} + p_2 \cdot QoS_{S_3}) \cdot \prod_{\{S_1, S_4\}} QoS_{S_i}$	${QoS_{S_1}}^n$
Additive QoS	$\sum_{\{S_1 \sim S_4\}} QoS_{S_i}$	$\sum_{\{S_1, S_4\}} QoS_{S_i} + \max_{\{S_2, S_3\}} QoS_{S_i}$	$p_1 \cdot QoS_{S_2} + p_2 \cdot QoS_{S_3} + \sum_{\{S_1, S_4\}} QoS_{S_i}$	$n \cdot QoS_{S_1}$

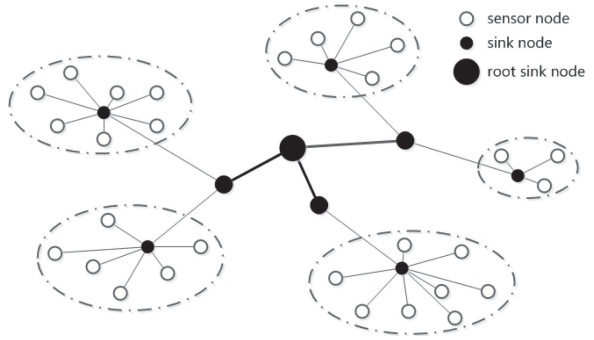


Fig. 3. The network structure.

For the energy model, we consider power for sensing, power for receiving and power for transmitting. The energy consumption formulas we use throughout this paper are as follows:

$$P_{Sense} = \alpha_1 b,$$
$$P_{Tx} = (\beta_1 + \beta_2 r^n) b,$$
$$P_{Rx} = \gamma_1 b,$$

where b (in bits/sec) is individual sensor node's data rate. The term r^n accounts for the path loss, and the typical value for n is 2 or 4. According to [16], some typical values for the parameters are as follows:

$$\alpha_1 = 60 \times 10^{-9} J/bit,$$
$$\beta_1 = 45 \times 10^{-9} J/bit,$$
$$\beta_2 = 10 \times 10^{-12} J/bit/m^2 \ (when\ n = 2),$$
$$\text{or}, \beta_2 = 0.001 \times 10^{-12} J/bit/m^4 \ (when\ n = 4),$$
$$\gamma_1 = 135 \times 10^{-9} J/bit.$$

2.1.3 Optimal Service Selection in WSNs

Based on a user's functional requirements, an abstract workflow which consists of a set of abstract services will be constructed. Here, each abstract service is associated with a set of pre-designed concrete atomic services with the same function. Furthermore, based on a user's *QoS* requirements, a concrete atomic service selection process for each abstract service should be done.

Definition 1 (Feasible Service Selection). *For a given abstract workflow $AW = AS_1, AS_2, \ldots, AS_m$ and a vector of global* QoS *constraints $C = C_1, C_2, C_3, \ldots, C_h$, we consider a selection of concrete services to be a feasible selection, if it contains exactly one concrete service for each abstract service appearing in AW and its aggregated* QoS *values satisfy the global* QoS *constraints.*

In resource constrained *WSN*s, the effect of residual energy on atomic service selection should be considered. Therefore, it is necessary to take consideration of both energy efficiency and *QoS* performance. That is to say, we will ensure the *QoS* constraints guaranteed in terms of a suboptimum way, and within the available power budget.

Definition 2 (Optimal Service Selection in WSNs). *In resource constrained* WSN*s, for a given abstract workflow $AW = AS_1, AS_2, \ldots, AS_m$ and a vector of global* QoS *constraints $C = C_1, C_2, C_3, \ldots, C_h$, we consider* optimal selection *to be the feasible selection (see Definition 1) which maximizes the overall network lifetime.*

2.2 Related Work

In order to select the optimal services that satisfy a user's *QoS* constraints, many efforts have been made. A bulk of related work can be mainly divided into the *local QoS optimization*, the *global QoS optimization* and the *approximate* approach.

The *local QoS optimization* [17], [18] usually applied two phases of normalization. The first one aims to measure *QoS* attributes independent of units. While in the second one, each *QoS* attribute is assigned with a weight, and the utility value will be calculated by Simple Additive Weighting (*SAW*) [19]. Furthermore, the service with largest utility will be selected. However, service selection is a constrained multi-objective problem. The *local QoS optimization* approach selects the optimal service for each abstract service independently, and cannot make sure that the composed workflow is *QoS* guaranteed [17]. Hence, Yu et al. [13] modeled the service selection in two ways: a multidimension multichoice 0-1 knapsack (MMKP) problem and a multiconstraint optimal path (MCOP) problem. By solving these two problems, the optimal service can be selected. Ardagna et al. [20] formalized service selection as a mixed integer programming (*MIP*) problem. Then negotiation techniques were exploited to identify a feasible solution. Due to the *global QoS optimization* approach has to traverse all the services which is NP-Hard, Qi et al. [21] utilized the dependence membership among services to filter the impossible ones, in order to decrease the scale of candidate services. Li et al. [22] utilized trust-based method to filter out untrustworthy services. Canfora et al. [23] proposed a lightweight approach that uses genetic algorithms for the optimal service selection. Myoung et al. [24] proposed a constraint satisfaction based algorithm that combines tabu search and simulated annealing metaheuristics. In addition, some research [11] turns to the *approximate* approach, which could get the suboptimal service with lower computation complexity.

However, *WSN*s are resource constrained, with lower computation capability, less storage space, limited communication bandwidth and especially limited energy. Hence, energy-efficient service selection and the corresponding adaptation is critical in *WSN*s and has been the aim of many research efforts. Jayapal et al. [25] proposed an adaptive service selection protocol. The service selection protocol is based on both the distance between the service requestor and the service provider and the remaining lifetime of the service provider. Zhang et al. [26] proposed a cross-layer approach to select a service provider. They select services according to a cost function which depends on the service information such as the response time and the battery level. Chien-Liang Fok et al. [27] presented *Adaptive Servilla*, which is a middleware that provided adaptive service selection capabilities to coordinate the resources used by *WSN* applications. Based on the middleware, they proposed the energy efficient service selection mechanism. In addition, they indicated that service sharing can be integrated into the middleware, if the results of one service also satisfy multiple users.

Indeed, traditional task/resource scheduling mechanisms in *WSN*s succeed to realize the energy-efficient node selection or task allocation. A strategy for energy saving is

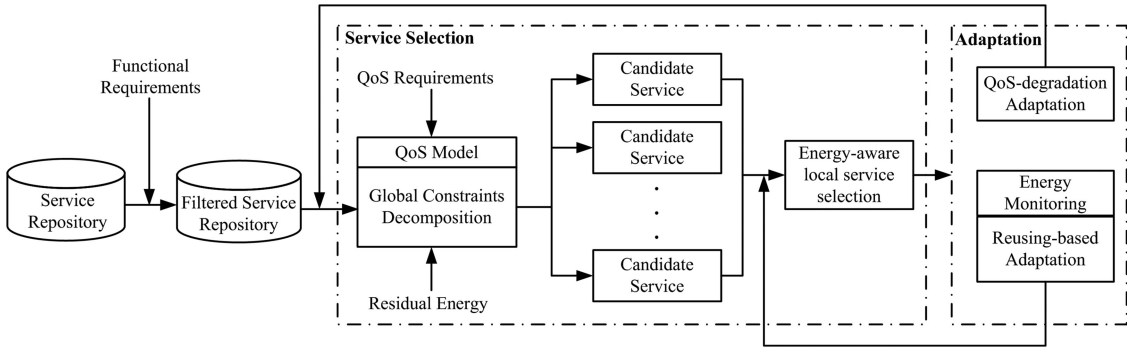


Fig. 4. The flowchart for the proposed energy-aware service selection.

to cleverly manage the duty cycle of sensors. Delicato et al. [28] formalize the node selection problem as a knapsack problem and adopt a greedy heuristic to maximize residual energy of selected nodes to extend the network lifetime. In [29], authors indicated that one application's operations can be decomposed into tasks to be allocated to each infrastructure component. Considering the energy cost of each task, they proposed a lifetime maximization algorithm based on an iterative and asynchronous local optimization of the task allocations between neighboring nodes. Although the above approaches accomplished the energy-efficient service selection/task allocation in *WSN*s, which can bring balanced energy consumption, they neglected to consider multi-*QoS* constraints. Then, Li et al. [30] proposed a mechanism that dynamically coordinate the sharing of the available resources to optimize resource utilization while meeting application requirements. However, they failed to consider the workflow-based application in *WSN*s.

Although a number of efforts have been done on energy-efficient service selection or task/resource allocation, they all failed to resolve the simultaneous multi-constraints (users' multi-*QoS* constraints) and workflow-based (a *WSN* application consists of a list of sequential subtasks) service selection problem. In order to resolve this problem, workflow-based service selection with multi-*QoS* constraints should be figured out and the network lifetime should also be extended as long as possible. In the adaptation mechanism, existing research also failed to consider the multi-user service selection scenario, which can also be studied to improve energy efficiency. Hence, existing research can not be directly applied to the application scenario considered in this paper. Therefore, in this paper, we will combine energy efficiency and *QoS* constraints and propose an energy-efficient and *QoS* oriented service selection and adaptation mechanism.

3 QoS-Guaranteed Energy-Efficiency Atomic Service Selection

Traditional service selection approaches try to achieve the best *QoS* and fail to implement the balanced energy consumption. Hence, in this paper, we will propose a *QoS*-guaranteed energy-efficient service selection for resource constrained *WSN*s. If we adopt the global service selection approaches, we can consider the factor of service energy and get the optimal w ($w \geqslant 1$) candidate services. But, the computation complexity is high and we have to re-run the entire procedure when adaptation. If we adopt local service selection approaches, the global *QoS* constraints may not be satisfied. Therefore, we will combine the global service selection and the local service selection, in order to achieve optimal *QoS* guaranteed service selection with low computation complexity in *WSN*s.

It is worth to note that, the service selection mechanism should consider the effect coming from the underlying sensor network, such as topology, communication and energy model. Take the service selection for example, when we select one service (i.e., S_1), we should not only consider the state of S_1, but also consider the state of the corresponding relay nodes. However, the main contribution of this paper is to discuss the residual energy aware service selection and adaptation. Hence, for simplicity, we also assume that each sensor node in the network is deployed with only one service. Under the above assumptions, we will discuss the service selection and adaptation mechanism.

As shown in Fig. 4, our proposed approach can be mainly divided into four steps: *functionality-based service filtering*, *global QoS constraints decomposition*, *energy-aware local service selection* and *dynamic workflow adaptation*. First, a filtering process will be done to filter out the services with unsatisfied function according to a users' functional requirements; Second, we decompose the global *QoS* constraints (for a workflow) into a set of local *QoS* constraints (each local *QoS* constraints item for an abstract service); Then, based on the set of local *QoS* constraints, the optimal service for each abstract service will be selected by considering both *QoS* and energy. Finally, the service energy will decrease along with the service running. Hence, we will also monitor the residual energy at runtime and invoke the service reselection process when the specified conditions are satisfied. The detailed description of the modules in Fig. 4 will be represented in the following sections.

3.1 Candidate Service Clustering

In order to accomplish the *QoS* decomposition, we will first perform a clustering process for each abstract service to organize the large number of concrete atomic services. First, we can run the global optimal service selection based on clustered services but a single service. Then, a local optimal service selection will be performed in each service cluster. In this way, we can achieve optimal *QoS* guaranteed service selection with low computation complexity. Among many different clustering algorithms, *K-means* is a simple one to cluster a data set with a pre-determined block number *K*. Hence, we will use *K-means* to get atomic service clusters. In

this paper, we use the *Euclidean distance* to measure the distance between each two atomic services.

In order to avoid comparing different *QoS* attributes with very different value ranges, a normalization procedure will be done.

$$QoS'_{i,j}(S) = \begin{cases} \frac{QoS_{i,j}(S) - QoS_i\_min}{QoS_i\_max - QoS_i\_min}, if\ QoS_{i,j}(S)\ is\ positive; \\ \frac{QoS_i\_max - QoS_{i,j}(S)}{QoS_i\_max - QoS_i\_min}, if\ QoS_{i,j}(S)\ is\ negative; \end{cases} \tag{4}$$

where $QoS_{i,j}(S)$ means the value of the jth *QoS* attribute of concrete service S in abstract service AS_i, QoS_i_min represents the minimize value of the jth *QoS* attribute value in AS_i, and QoS_i_max represents the maximize value of the jth *QoS* attribute value in AS_i. A positive *QoS* means that the larger the value is, the better the *QoS* (e.g., availability), while a negative *QoS* means that the larger the value is, the worse the *QoS* (e.g., response time). $QoS'_{i,j}(S)$ is the normalized $QoS_{i,j}(S)$, and $0 \leq QoS'_{i,j}(S) \leq 1$.

Suppose we have two atomic services S_a and S_b, the distance between S_a and S_b can be computed as

$$Dist(S_a, S_b) = \frac{1}{h} \times \sqrt{\sum_{i=1}^{h} (QoS'_{i,j}(S_a) - QoS'_{i,j}(S_b))^2}, \tag{5}$$

where h is the number of *QoS* attributes. The clustering algorithm is shown in Algorithm 1. Here, the value of K is set to 10. In fact, the optimal value of K is related to the distribution of the atomic service *QoS*.

Algorithm 1. Atomic Service Clustering Algorithm

Require: atomic service set S, the *QoS* attribute values of each atomic service, and the residual energy of each atomic services.
Ensure: atomic service clusters.
1: select K atomic services from S;
2: Set each selected atomic service s_i as centroid of a cluster c_i;
3: **repeat**
4: **for** $s_i \in S$ **do**
5: $maxDist = 0$;
6: $cIndicator = 0$;
7: **for** $c_j \in C$ **do**
8: $centroid = Centoid(c_j)$;
9: **if** $maxDist < Dist(s_i, centroid)$ **then**
10: $maxDist = Dist(s_i, centroid)$;
11: $cIndicator = c_j$;
12: **end if**
13: **end for**
14: assign s_i to the cluster $cIndicator$;
15: **end for**
16: **for** $c_i \in C$ **do**
17: update the cluster centroid of $c_i = (\frac{\sum QoS_{q,1}(s)' \in c_i}{|c_i|}, \frac{\sum QoS'_{q,2} \in c_i}{|c_i|}, \ldots, \frac{\sum QoS'_{q,h} \in c_i}{|c_i|}, \frac{\sum Energy(s) \in c_i}{|c_i|})$;
18: **end for**
19: **until** no change in clusters C
20: Return K atomic service clusters;

3.2 Decomposition of Global *QoS* Constraints

After the clustering process, we can get K service clusters ($c_1, c_2, \ldots, c_k$) and corresponding K cluster centroids ($c_1.centroid, c_2.centroid, \ldots, c_k.centroid$) for each abstract service. Obviously, these K cluster centroids can represent the service set in their clusters. Hence, based on these K cluster centroids, we can divide the quality range of each *QoS* attribute into a set of discrete quality values, which are called quality levels. In this paper, the jth quality level for the ith abstract service is set to

$$\mathcal{QL}_{i,j} = \frac{1}{2} \times \{\mathcal{QoS}_i(c_j.centroid) + \mathcal{QoS}_i(c_{j+1}.centroid)\} \tag{6}$$

Here, $\mathcal{QoS}_i(c_j.centroid)$ is a *QoS* attributes vector and can be represented as $\{QoS_i^1(c_j.centroid), QoS_i^2(c_j.centroid), \cdots, QoS_i^h(c_j.centroid)\}$, where h is the number of *QoS* attributes. In this way, we can get $K-1$ discrete quality levels for each abstract service. Then, these quality levels will be used as candidate local constraints in the local service selection algorithm.

The most important issue of service selection is to find the service that satisfies a user's *QoS* requirements. However, in practical applications, users have different preferences on different *QoS* attributes. Therefore, we utilize a preference matrix $\mathcal{P}$ to represent a user's preferences:

$$\mathcal{P} = (\ p_1 \quad p_2 \quad \cdots \quad p_h\ )^T$$

Then we can get the utility of an atomic service:

$$Utility(AS_{i,j}) = \mathcal{Q}_i \cdot \mathcal{P} = \sum_{\theta=1}^{h} QoS_{i,j}^{\theta} \cdot p_\theta \tag{7}$$

Here, utility indicates a user's satisfaction for *QoS*. The larger the utility is, the better the service may satisfy a user's *QoS* requirements.

We assign a weight for each abstract service. An abstract service with higher energy will be assigned with a smaller weight, vice versa. In this way, the abstract service with lower energy will have loose local *QoS* constraints. Hence, there will be more concrete services to perform the balanced energy consumption.

$$w_i = \frac{\overline{E}_{max} - \overline{E}_{AS_i}}{\overline{E}_{max} - \overline{E}_{min}} \tag{8}$$

In which, w_i is the weight of the ith abstract service AS_i; $\overline{E}_{AS_i} = \frac{1}{N_i} \cdot \sum_{j=1}^{N_i} E_{AS_{i,j}}$. It represents the average energy of all the services in AS_i; $\overline{E}_{max}$ is the maximum average energy among all the abstract services; $\overline{E}_{min}$ is the minimum average energy among all the abstract services. Hence, in order to achieve the optimal decomposition, we have the following assumptions,

- the best decomposition plan should make sure the local *QoS* constraints can cover as many candidate services as possible.
- the best decomposition plan should have the best utility.

Based on the above two assumptions, we then assign each quality level $QL_{i,j}$ with a value z_i between 0 and 1, which estimates the benefit of using this quality level as a local constraint. This value is calculated as follows. First, we compute $N(QL_{i,j})$ (i.e., the number of candidate services that would qualify if level $QL_{i,j}$ is used as the local constraint). Second, we calculate the utility of each candidate

service in the service class using the utility function in Formula (7) and then get $U(QL_{i,j})$ (i.e., the average utility value that can be obtained by considering these qualified services). Finally, in order to evaluate the quality of decomposition plan, z_i can be calculated as

$$z_i = w_i \cdot \frac{N(QL_{i,j})}{N(total)} \times \frac{U(QL_{i,j})}{U(max)}, \tag{9}$$

where $N(total)$ means the total number of atomic services in an abstract service, while $U(max)$ means the available highest utility in an abstract service.

Definition 3 (Optimal Decomposition). *For a given set of quality levels and a vector of global* QoS *constraints* $C = C_1, C_2, C_3, \ldots C_h$, *we consider* optimal decomposition *to be the feasible selection (see Definition 1) that maximizes the overall* z_i.

We use mixed integer program (*MIP*) model to find the best decomposition of global *QoS* constraints into local *QoS* constraints. We use a binary decision variable $x_{i,j}$ for each local quality level $QL_{i,j}$ such that $x_{i,j} = 1$ if $QL_{i,j}$ is selected as a local constraint for abstract service AS_i , and $x_{i,j} = 0$ otherwise. Therefore, we use the following allocation constraints in the model:

$$\forall i, \sum_{j=1}^{k-1} x_{i,j} = 1,\ 1 \leqslant i \leqslant m. \tag{10}$$

The objective function of our *MIP* model is to maximize the overall z value (as defined in Formula. 9). Therefore, the objective function can be expressed as follows:

$$maximize \sum_{i=1}^{m} \sum_{j=1}^{k-1} z_{i,j} \cdot x_{i,j}. \tag{11}$$

The selection of the local constraints must ensure that global constraints are still satisfied. Therefore, we add the following set of constraints to the model:

$$\forall \theta, \sum_{i=1}^{m} \sum_{j=1}^{k-1} QoS_{i,j}^{\theta} \cdot x_{i,j} < C_i,\ 1 \leqslant \theta \leqslant h. \tag{12}$$

By solving this model using any *MIP* solver methods, we get a set of local quality levels. These quality levels are then sent to the distributed set of involved service brokers to perform local selection.

3.3 Local Service Selection

After getting the optimal quality level (i.e., local *QoS* constraints) for each abstract service, we can perform local service selection for each abstract service independently. Our goal is to satisfy a user's *QoS* requirements better, and prolong the network lifetime as long as possible.

It is worth to note that, traditional services are responsible for the execution of specific tasks, such as computing. Overmany requests will result in the overload of services. Hence, load balance scheme should be done among services. Unlike traditional services, services in *SOA*-oriented *WSN*s are most data services, which respond to gathering the environment information. When a service is running, another request for this service will just get the copy of previous sensed data and will not bring additional burden. Through service reusing, we can decrease the number of activated services and furthermore largely decrease the energy consumption.

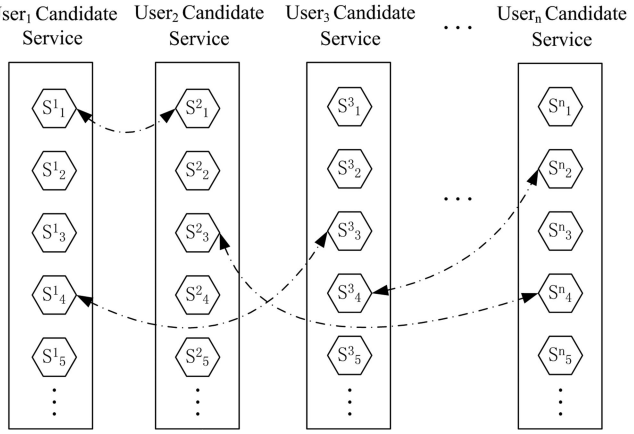


Fig. 5. The service selection based on service reusing.

To sum up, in resource constrained *WSN*s, local service selection should consider the following issues,

- the *QoS* of the candidate service.
- the residual energy of the candidate service.
- the running state of the candidate service.

As shown in Fig. 5, S_j^i represents the jth candidate service of User i. The dashed line indicates that the linked two services are indeed the same service. For example, $S_1^1 = S_1^2$, $S_4^1 = S_3^3$, $S_3^2 = S_4^n$ and $S_4^3 = S_2^n$.

For each group of candidate services, we consider both *QoS* and energy.

$$\forall i, \exists j, maximize \Big\{ \lambda_1 \cdot \sum_{\theta=1}^{h} QoS_\theta(S_{i,j}) \cdot p_\theta(S_{i,j}) + \lambda_2 \cdot E_{S_{i,j}} + \lambda_3 \cdot \Delta E_{S_{i,j}} \Big\}, 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n, \lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{13}$$

$S_{i,j}$ means the jth concrete service of the ith abstract service; $QoS_\theta(S_{i,j})$ represents the θth *QoS* attribute of $S_{i,j}$; $E_{S_{i,j}}$ is the residual energy of $S_{i,j}$; $\Delta E_{S_{i,j}}$ is the energy consumption rate of $S_{i,j}$; $\lambda_{(1/2/3)} \in [0, 1]$ is the weight. All the *QoS* and *energy* attributes should be normalized to [0,1]. For the applications which are sensitive to *QoS*, λ_1 can be set to a larger one, vice versa.

There may exist the situation that one atomic service is much better in *QoS*, but worse in *energy*. According to Formula (13), this atomic service will be still selected, though it is nearly energy exhausted. In order to address this issue, we adopt the best-R method. We will first select the best R atomic services according to Formula (13). Then, two cases will be considered. The first one is that some of the R services are being used by other users. In this case, we will select the services which are in the state of running and own the best overall performance on *QoS* and energy. The second one is that all the R services are idle. In this case, we will furthermore select the atomic service that has the best energy among these R atomic services.

4 QoS-Guaranteed Energy-Efficiency Dynamic Workflow Adaptation

By reusing the running service, the proposed service selection approach can avoid the unnecessary energy consumption. However, with the increased service requests,

the existing service selection plan may not be the optimal one and adaptation should be activated in order to balance the energy consumption. For example, as shown in Fig. 5, suppose $User_1$ select S_1^1 according to Formula (13). Apparently, $User_2$ will select S_1^2. In this case, if $User_3$ select S_3^3, there will be a decision problem as S_4^1 and S_3^3 are the same service. There will be two service selection plans: (1) S_1^1, S_1^2 and S_3^3; (2) S_4^1, S_X^2 and S_3^3. Here, S_X^2 will be specified according to Formula (13). However, if $User_3$ does not select S_3^3, the above adaptation will not happen. Furthermore, long time running of S_1^1 (i.e., S_1^2) will consume its energy quickly. Hence, changing the component services of a workflow at runtime according to each component service's residual energy is necessary. However, frequently changing the execution process of a workflow will also bring great computation cost and result in low performance. Therefore, a mechanism to trade-off the cost of the execution process changing and the network lifetime should be developed.

4.1 QoS Degradation Adaptation

During the workflow running process, *QoS* may change and no longer satisfy users' *QoS* constraints. Hence, *QoS monitoring* should be proposed. *QoS monitoring* can be invoked in two ways: periodically execution and passive invocation upon a user's feedback. The purpose of service selection is to satisfy a user's functional requirements with acceptable *QoS*. Hence, a *QoS* guarantee mechanism is important for the successful execution of a workflow. In the case of event-driven applications, such as *intrusion detection*, users are usually unaware when the application *QoS* is not guaranteed. Also, for less *QoS* sensitive applications (e.g., temperature control), users will not sense the *QoS* degradation quickly. Last but not least, there exists the problem of user's reputation. Hence, users' feedback is insufficient, and periodical *QoS* monitoring should be used to verify whether the *QoS* degradation occurs or not. In this paper, we adopt our proposed *QoS degree* based *QoS* monitoring in [31] and perform the prediction of *QoS* degradation with a regression model. On one hand, the *QoS* monitoring processes are only performed on the activated services. On the other hand, we proposed the *QoS* degree based *QoS* monitoring. The monitoring frequency closely depends on the fluctuation of the *QoS* degree. When the *QoS* is going to degrade, we will receive more *QoS* information to predict possible degradation before it real happens. In most cases, *QoS* degree is high and we will adopt a longer monitoring period to bring lower additional burden.

When *QoS* degradation happens, the previous global *QoS* decomposition plan may not be optimal. Then, the adaptation process will restart from *candidate service clustering*, and perform *global QoS constraints decomposition* and *local service selection*.

4.2 Energy-Based Adaptation

The services we mentioned are all long running services. During the running process, the energy of the selected service will be decreasing. Hence, an efficient approach should be proposed to balance the energy consumption among different services.

In the service running process, we will monitor the residual energy of each running service and its corresponding $R-1$ services. The invoke condition of energy-aware adaptation is:

$$S' < \tau \cdot \frac{1}{R-1} \cdot \sum_{i=1}^{R-1} E_{S_i}. \tag{14}$$

As shown in Formula. 14, for each running service, we first compute the average residual energy of the other R-1 services. In this paper, τ is set to 80 percent. When the energy of the running service is lower that the 80 percent of the average energy, the adaptation program will be activated to re-select the optimal service according to Formula (13).

4.3 Reusing-Based Adaptation

For a specific abstract service, different users may have different *QoS* constraints. Then, different users will have different candidate services. Reusing-based adaptation will occur when more than one user uses the same abstract service and their candidate service sets have intersection. Take a simple case for example, service S_3^3 is selected by $User_3$. Obviously, S_3^3 is also the candidate service of $User_1$, i.e., the candidate service sets of $User_1$ and $User_3$ have an intersection. In this case, an evaluation program will be activated to decide whether to start the reusing-based adaptation or not. Reusing-based adaptation will occur when more than one user uses the same abstract service and their candidate service sets have an intersection. Hence, in order to perform reusing-based adaptation, we should first check if a service exists in other users' candidate service set and furthermore get the current user-service mapping relationship (i.e., which user uses which services).

4.3.1 Service Membership Query

Bloom filter (*BF*), conceived by Burton Howard Bloom [32] in 1970, is a hashing-based data structure that succinctly represents a set of elements to support membership queries. Due to its temporal and spatial efficiency, *BF* can be utilized in our service membership query.

The *BF* initializes a bit array with the size of m, which are all initialized to 0. For simplicity, we use $B[m]$ to represent the *BF*. It uses ϕ independent hash functions to hash an element into ϕ of m array positions. We will exploit the *BF* to manage sensor nodes. We regard each node's attribute as an element represented as simply A. When inserting an element A_i into the bit array, the element A_i will be hashed by ϕ hash functions to get array positions $h_1(A_i)$, $h_2(A_i)$ ..., $h_\phi(A_i)$. The corresponding positions in the bit array will be set to 1. Here, the position can be set to 1 multiple times, but only the first change takes effect.

Definition 4 (Membership). *Let $\mathcal{A}'$ be the object to execute a membership query. Given ϕ independent hash functions and a Bloom filter $\mathcal{B}[m]$, $\mathcal{A}'$ is the member of $\mathcal{B}[m]$ if and only if all the hashed positions $h_1(\mathcal{A}'), h_2(\mathcal{A}'), \ldots, h_\phi(\mathcal{A}')$ in $\mathcal{B}[m]$ are set to 1.*

As shown in Fig. 6, we assume $\phi = 3$. As $B[h_1(\mathcal{A}'_2)] = B[h_2(\mathcal{A}'_2)] = B[h_3(\mathcal{A}'_2)] = 1$, according to Definition 4, we know that $\mathcal{A}'_2$ is the member of $\mathcal{B}[m]$. In the same way, due to $B[h_1(\mathcal{A}'_1)] = B[h_2(\mathcal{A}'_1)] = 0$, $\mathcal{A}'_1$ is not the member of $\mathcal{B}[m]$.

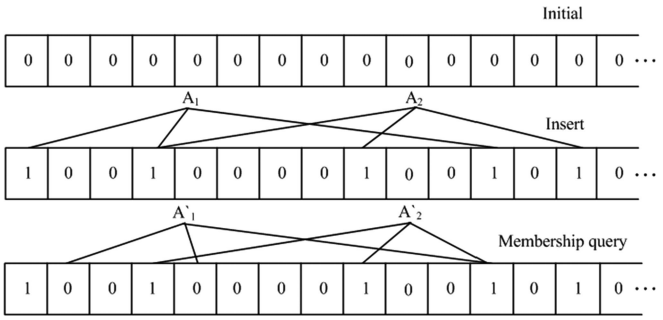


Fig. 6. Standard Bloom Filter.

By utilizing bloom filter, we can achieve the service membership query quickly. In order to utilize *BF*, we assign each service a unique ID. By hashing a service ID, we can insert the service into *BF*. Then, for all candidate services, we construct a *BF*, as shown in Fig. 7. Compared to the standard *BF*, each entry will consist of two parts: a bit and a connected linked list. This design cannot only indicate the status of service insertions, but also allow a pointer to connect associated linked list for further string parsing and service ascription. Suppose $\phi = 3$, the service storage process is described as follows. $User_1$'s candidate services contain $service_1$, "$Service_1.ID$" will be hashed into three entries b_8, b_{12} and b_{15} respectively. In addition to setting the corresponding entries to 1, the connected linked list of the first hashed entry b_8 will be added with a string "$User_1\#Service_1$". Then $User_2$'s $Service_1$ comes, '$Service_1.ID$" will be hashed into the same entries as the ones of the first service. While the "$User_2\#Service_1$" will be added into the first shortest linked-list L_{12} connected b_{12} for global storage balance of linked lists. In a membership query, the service's ID will be hashed and the connected three linked lists will be received. From these linked lists, we can get the user set who owns the certain service by parsing the corresponding strings.

4.3.2 Adaptation Mechanism

In reusing-based service selection, the new service selection may affect the existing service selection scheme as different users usually own the same services. Hence, in order to achieve the optimal service selection, we have the following optimized goals:

- The energy should be maximized. By utilizing the service with highest energy and lower energy consumption rate at all time, we can avoid some service exhausts its energy earlier.
- The number of services should be minimized. Smaller service number can bring much more reusing efficiency and reduce the overall energy consumption.
- Try to not change the existing service selection scheme. During the service running process, we should try to decrease the number of service adaptation in order to reduce the complicacy.

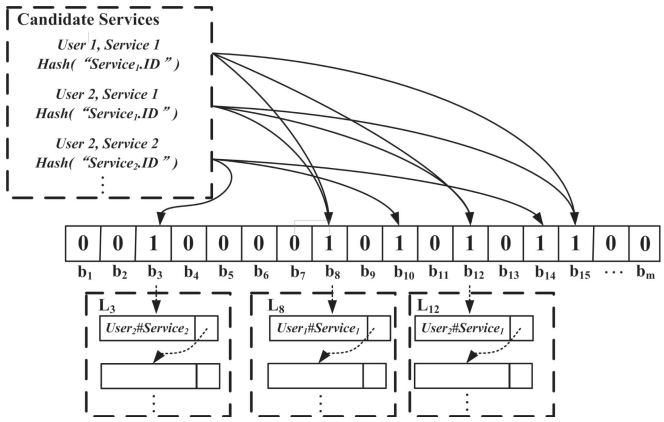


Fig. 7. Service Storage in Linkedlist-extended *BF*.

In deed, the above three goals may not be all satisfied. Our work is to try best to satisfy much more goals listed above. Suppose there are n users that request the same abstract service. Based on the n users, we have a group of service sets ($SS^1, SS^2, \ldots, SS^n$), in which SS^i corresponds to the candidate services of the ith user. We represent each service in all the service sets as a node. In addition, for arbitrary two service sets (SS^i and SS^{i-1}), there is an arc from SS^i_j $(j = 0, 1, \ldots, m)$ to SS^{i-1}_j $(j = 0, 1, \ldots, m)$. The weight of the arc can be computed as:

$$W^i_{j,\theta} = \begin{cases} 0, & if\ S^i_j\ and\ S^{i-1}_\theta\ are\ the\ same\ service; \\ \frac{1}{2} * \frac{E'^{i-1}_{max} - E'_{S^{i-1}_\theta}}{E^{i-1}_{max} - E^{i-1}_{min}}, & if\ S^{i-1}_\theta\ is\ the\ selected\ service\ currently; \\ 0, & if\ S^{i-1}_\theta\ has\ been\ already\ selected\ in\ the\ previous\ process; \\ 1 - \frac{1}{2} * \frac{E'_{S^{i-1}_\theta} - E'^{i-1}_{min}}{E^{i-1}_{max} - E^{i-1}_{min}}, & others; \end{cases} \quad (15)$$

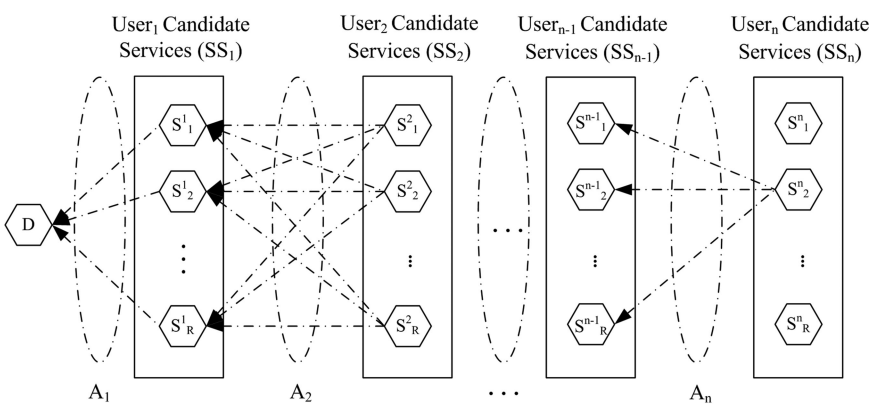


Fig. 8. The service adaptation based on service reusing.

Here, $W^i_{j,\theta}$ means the arc weight between S^i_j to S^{i-1}_θ; $E'_S = \lambda_1 \cdot E_S + \lambda_2 \cdot \Delta E_S$; $E'_{S^{i-1}_\theta}$ represents the energy of service S^{i-1}_θ; E^{ni-1}_{max} represents the maximum energy among the services in the $(i-1)$th service set; while E^{ni-1}_{max} represents the minimum energy among the services in the $(i-1)$th service set.

The overall model is shown in Fig. 8. As we can see, it is a multi-layer directed graph. The service adaptation problem can be represented as a single-pair (i.e., single source and single destination) shortest path problem in graph theory. It is worth to note that, D in Fig. 8 is only a fictitious node to indicate the destination and the weights of arcs directed to D are all 0.

Consider Fig. 8 as a directed graph (V, A) with source node S^n_2, target node D, and cost $w_{i,j}$ for each arc (i, j) in A. Then the resolution of the shortest path problem can be translated to the linear programming (*LP*) issue. This *LP* issue has the special property that it is integral. More specifically, every basic optimal solution (when one exists) has all variables equal to 0 or 1. Therefore, we use the following allocation constraints,

$$\begin{cases} \forall i, \sum_{j=1}^{R} x_{i,j} = 1, \ i = 1, n \\ \forall i, \sum_{j=1}^{R^2} x_{i,j} = 1, \ 2 \leqslant i \leqslant n-2. \end{cases} \tag{16}$$

Here, $x_{i,j} = 0$ or $x_{i,j} = 1$. $x_{i,j} = 1$ means that the jth arc in A_i has been selected, while $x_{i,j} = 0$ means that the jth arc in A_i has not been selected. Hence, Formula. 16 indicates that for any A_i, there is one and only one arc will be selected.

The object function can be expressed as follows:

$$minimize \sum_{i=1}^{n} \sum_{j=1}^{R} w_{i,j} \cdot x_{i,j}. \tag{17}$$

In Formula (17), $w_{i,j}$ represents the jth arc in A_i. After computing the minimum path from S^n_2 to D, the services on the path will be selected. Then, based on the newly selected services, adaptation will be achieved.

It is worth to note that, the layer of the model keep increasing with the number of users that request the same abstract service. However, it is complex and infeasible that the adaptation activated just when a new user try to request the same abstract service. The activation only happens when the services selected by the new user have an intersection with the currently running services set.

5 Experiments and Evaluations

By introducing the service energy to the service selection and workflow adaptation in resource constrained *WSN*s, we try to balance the energy consumption and network lifetime. To evaluate the capability of our proposed workflow management mechanism, a number of experiments on service selection and workflow adaptation are carried out in this section.

5.1 Experimental Settings

Before the experiments, we should first get enough dataset of services with corresponding *QoS* attributes. Some work [33], [34] has provided real datasets which are collected from real web services that exist on the Web. Unfortunately, the scale of these datasets is not large enough for our experiments. Hence, in our experiments, the datasets are created by assigning arbitrary *QoS* values. For simplicity, we use only three *QoS* attributes, i.e., *execution time*, *cost* and *availability*. Then, the values of *execution time*, *cost* and *availability* are distributed in the range between (60, 90), (120, 170) and (0.3, 1) respectively.

We utilize the open source Linear Programming (*LP*) solve system: lpsolve version 5.5 [35] for solving the corresponding programming problem in both approaches. Then, the evaluation will be divided into two parts. In order to simulate the service selection time efficiency in large scale service oriented *WSN*s, we perform the simulation in *Java*. In this simulation, we design a service selection case with m abstract services and n concrete services per abstract service. By varying the values of m and n, we will get a list of experimental cases. The other part of evaluation will be performed in *OPNET* in order to evaluate the *QoS* optimization and energy efficiency considering of *WSN* characteristics. In the *OPNET* experiments, we consider a stationary network with 40*40 nodes which are distributed uniformly over a planar square region with 200m * 200m dimensions. Without losing generalization, we assume the sink node is in the center of the sensing region. The sink node has a constant power supply and so, has no energy constraints. It can transmit with high power to all the nodes. Thus, there is no need for routing from the sink node to any specific node. For energy consumption, we used the energy model in Section 2.1.2. Each node has an initial energy of 2 joules. A node is considered non-functional if its energy level reaches zero. Packet lengths are 20 bit for data packets. The users' service requests follow the uniform distribution. That is to say, in every one second, there is one user request arrives. When the service selection procedure completed, the corresponding service execution will go on for 5 minutes. Each user will be assigned with its randomly generated *QoS* constraints and preferences.

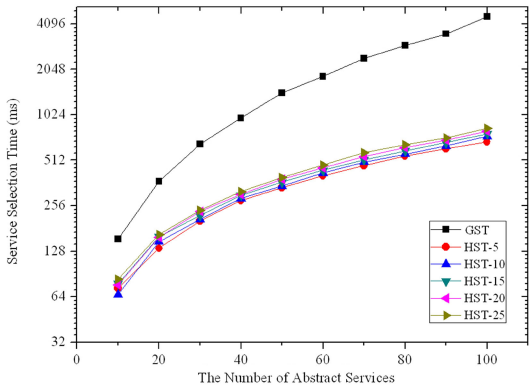
In order to select the optimal atomic services, one alternative approach is the *global optimization* approach, which will traverse all the possible atomic services. Hence, the service selection problem will be first solved by the traditional *global optimization* approach and then solved by our proposed *constraints decomposition* approach. Considering the factor of randomicity, all the experimental cases will be run 10 times and the average will be calculated.

5.2 Evaluation on the Service Selection Time Efficiency

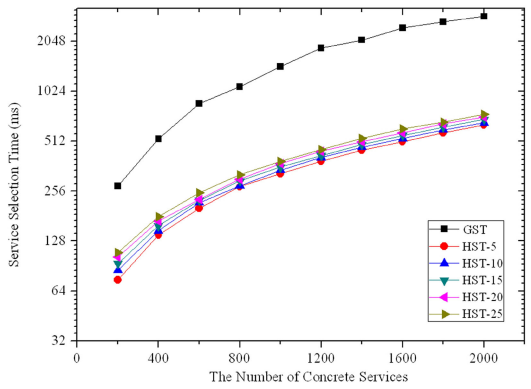
In this section, we will evaluate the time efficiency of the service selection process. The experiments will be carried out by investigating the performance of the proposed approach comparing with the *global optimization* approach.

Before the experiments, we analyze the time efficiency theoretically. Suppose there is an abstract workflow, which consists of n abstract services, l alternative concrete atomic services per abstract service and m global *QoS* constraints. Apparently, the computation complexity of *global optimization* approach is $O(n \cdot m \cdot l)$. Suppose the quality levels of our proposed approach is d, then the computation complexity of our approach is $O(n \cdot m \cdot d)$. If $d << l$, our approach has smaller computation cost than the *global optimization* approach.

Furthermore, by selecting different n, l, m and d, we did a set of experiments and recorded the average computation time. We performed this experiment in two cases: the first



(a) Different Abstract Services



(b) Different Concrete Services

Fig. 9. Service selection time comparison.

one is that n is set to 60 and l varies from 200 to 2000 with step 200, while the second one is that l is set to 1200 and n varies from 10 to 100 with step 10. In order to study the effect of the chosen number of quality levels in our proposed approach, we solved each experiment with different number of quality levels. Due to the value of K affects the performance of our proposed approach, we carried out these experimental cases with a set of different K values (i.e., 5, 10, 15, 20 and 25). For simplicity, we use *GST* to denote the time for global optimization approach, *HST* to denote the time for our proposed approach and *HST*-x denotes the service selection time for our proposed approach when the number of clusters is x.

The simulation results are shown in Fig. 9, where the service number is axis-*X* and the service selection time is axis-*Y*. Here, the service selection time contains the time consumed in *QoS* normalization and linear programming. As the service cluster in our proposed approach only executed once in the initial part, we eliminate the clustering time from the overall service selection time.

As we can see in Fig. 9, in both global optimization approach and our proposed approach, the service selection time increases with the number of abstract service. In addition, larger K results into longer service selection time.

5.3 Evaluation on the *QoS* Optimization

To implement the service selection, we adopt the *QoS* degradation approach. However, it has the disadvantage that the selected services may not be *QoS* optimized. In addition, we introduced the energy into the service selection, which may

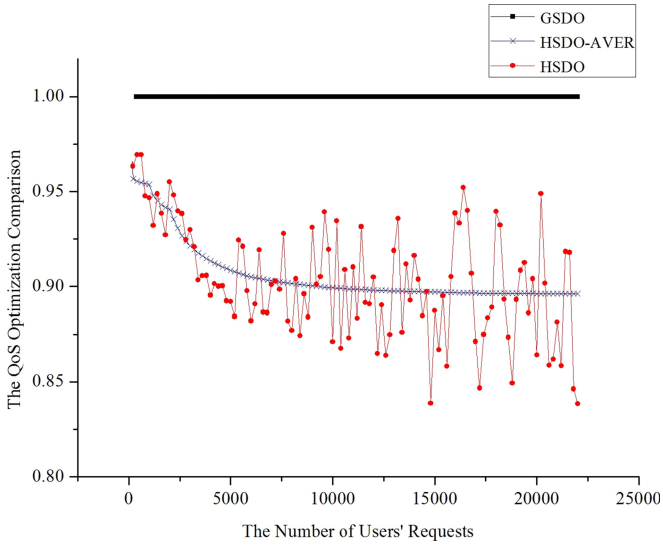


Fig. 10. The *QoS* optimization variation during the simulation process.

furthermore bring the *QoS* problem. Hence, in this section, we will evaluate the *QoS* optimization of the selected service by comparing it with the optimal service obtained by the global optimization approach. As the *QoS* attributes, users' preferences and *QoS* constraints are generated randomly, the optimal utility values are not the same for different service selection processes. This will bring unfair in comparison. Hence, in order to evaluate the *QoS* optimization, we set the utility of global optimization approach to 1 and the utility of our approach can be calculated as,

$$\widetilde{U}_{HS} = \frac{U_{HS}}{U_{GS}}. \tag{18}$$

Here, U_{GS} is the utility of global optimization approach, and U_{HS} is the utility of our approach. Then, $\widetilde{U}_{HS}$ is the normalized utility of our approach.

Take the service's energy into consideration, the simulation results are shown in Fig. 10. Here, we set $R = 5$ and $\lambda_1 = 0.3$, $\lambda_2 = 0.6$, $\lambda_3 = 0.1$. *GSDO* denotes the runtime *QoS* optimization of global optimization approach. In the simulation process, we get the runtime *QoS* optimization value every 100 user's requests and form the curve of *HSDO*. By calculating the average *QoS* optimization, we can get the curve of *HSDO-AVER*. As we can see, the curve of *HSDO* fluctuated during the simulation process. This is because we weight service's *QoS* and energy comprehensively in service selection mechanism. Services with better *QoS* will be priority selected, which consumes their energy heavily. Some time later, services with worse *QoS* but higher energy will replace the previous services to be selected, which bring lower *QoS* optimization. This case happens to and fro and brings the fluctuation of *QoS* optimization. However, the curve of *HSDO-AVER* keeps descending. That is to say, by considering the energy information, our selected services's *QoS* optimization keep descending with the energy consumption. But, the average *QoS* optimization degree is also higher than 89 percent, which is acceptable.

5.4 Evaluation on the Network Lifetime

In addition to the above results, we also evaluate whether our proposed approach can prolong the network lifetime

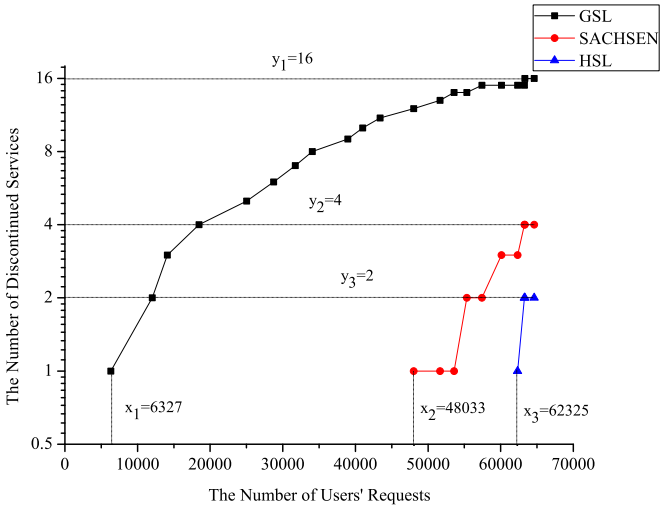


Fig. 11. The number of discontinued services comparison during the simulation process.

compared with the traditional workflow management approach and *SACHSEN* approach [30]. Before the experiments, the definition of network lifetime should be unified. In this paper, we adopt a simple but general definition of the network lifetime, which is the time duration from the time when the network starts running to the time when any discontinued service occurs due to energy exhausted.

When selecting the optimal atomic service, our approach takes into consideration of both *QoS* constraints and service energy. Hence, our proposed approach has better energy efficiency and furthermore longer network lifetime, which makes it suitable for resource-constrained *WSN*s. Then, we will make the qualitative analysis about the variation of the energy. The experiment is run for 3000s and simulation results are shown in Figs. 11 and 12. *GSL* denotes the lifetime of global optimization approach, *SACHSEN* denotes the lifetime of *SACHSEN* approach and *HSL* denotes the lifetime of our proposed approach.

As we can see in Fig. 11, our proposed approach extends the network lifetime comparing with the global optimization approach. After 6327 requests, there is one service exhaust its energy and dead in global optimization approach. The first discontinued service in *SACHSEN* approach appears after 48033 requests. While, the first discontinued service in our proposed approach appears after 62325 requests. When the simulation ends, the number of discontinued services is 16 in the global optimization approach and 4 in *SACHSEN* approach, larger than the number 2 in the proposed approach.

As shown in Fig. 12, X axis and Y axis have no physical significance. Axis-X varies from 1 to N (the number of concrete services of each abstract service), while axis-Y varies from 1 to M (the number of abstract services). Each point represents one service and points with darker color have lower residual energy, vice versa. As we can see in Fig. 12a, in the global optimization approach, the service execution happens on a fraction of total services. Then, these services consume energy heavily while others still have subtotal energy. In *SACHSEN* approach, the unbalanced energy consumption has a certain degree of improvement. However, in our proposed approach, the overall energy consumption was distributed among most of the total services, as shown in Fig. 12c.

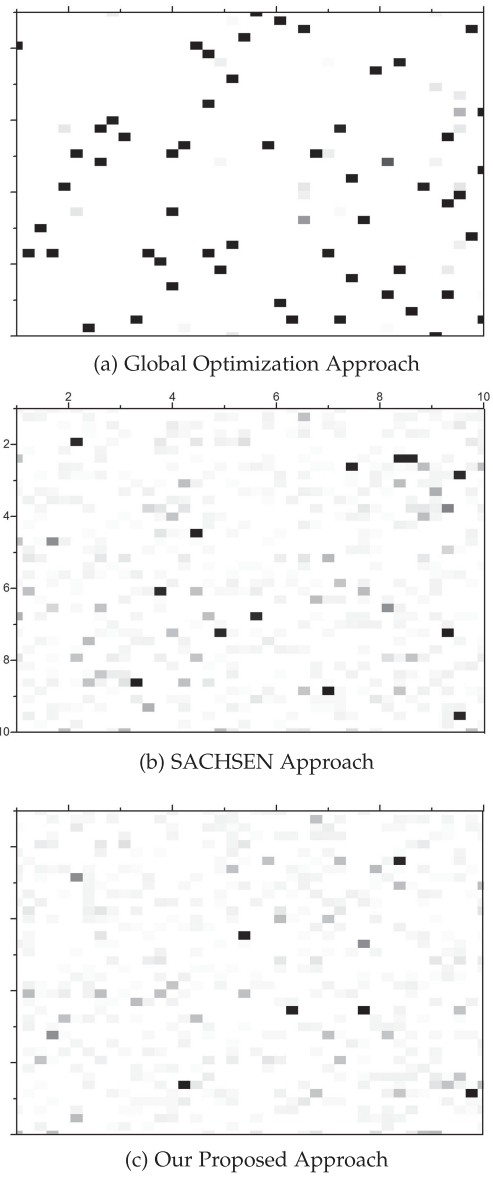


Fig. 12. Residual energy distribution and variation of proposed approach compared with SACHSEN approach and global optimization approach

Just because of this, our proposed approach can avoid energy hotspots and furthermore prolong the network lifetime.

5.5 Discussion

From the above experiments, we can see that our proposed workflow management mechanism have the important characteristic of energy efficiency, which is more suitable for resource-constrained *WSN*. On one hand, the proposed approach adopts the *QoS* decomposition mechanism, which decrease the time consumed for service selection with little loss in *QoS* optimization. On the other hand, the proposed approach takes both the service *QoS* and energy into consideration. By realizing the balanced energy consumption among services, the network lifetime was prolonged.

In order to achieve better performance, some parameters should be considered carefully. In the *QoS* decomposition process, different *QoS* attributes distribution makes the general suitable value selection for K not an easy task. In addition, within the service selection process, λ should be

chosen carefully as a trade off between the *QoS* optimization and the network lifetime. Finally, the value of τ in Formula (14) should also be set carefully. Larger τ will bring lower adaptation frequency, which may result in that some services (especially the services with better *QoS*) consume energy faster. When users' requests with higher *QoS* requirements arrives, services with better *QoS* will be activated inevitably, and then exhaust energy earlier. While smaller τ will bring higher adaptation frequency, which will increase system complexity, meanwhile, frequent adaptation also bring unnecessary energy consumption.

6 Conclusions

Workflow management in resource-constrained *WSN*s is an important issue which should be paid much more attention. In this paper, we introduce the service energy to workflow management in *WSN*s. First, we give the computation method of service energy and extend the traditional *QoS* model. Then, we adopt a *QoS* constraint decomposition approach. By considering both service *QoS* and service energy, global *QoS* constraints for workflow will be decomposed into a set of local *QoS* constraints for atomic services. Finally, efficient local service selection will be achieved. It is worth to note that, a service's energy will be consumed continually in the running process. In addition, with the increase of service requests, existing service selection scheme is not appropriate. Hence, we propose a dynamic workflow adaptation mechanism, which consists of the proposed service reusing and improved bloom filter. Compared with existing work on workflow management, the proposed approach implements the balanced energy consumption which is suitable for resource-constrained *WSN*s.

Results from experiments indicate the proposed service selection approach is efficient in time compared with the traditional global optimization approach and task allocation approach. In addition, the proposed adaptation implements the balanced energy consumption and longer network lifetime.

Acknowledgments

This research is supported by The National Key Research and Development Program of China (2017YFB0203501), Beijing Science and Technology Project (Z171100001117147).

References

[1] M. H. Valipour, B. AmirZafari, K. N. Maleki, and N. Daneshpour, "A brief survey of software architecture concepts and service oriented architecture," in *Proc. 2nd IEEE Int. Conf. Comput. Sci. Inform. Technol.*, 2009, pp. 34–38.

[2] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*. Hoboken, NJ, USA: Wiley, 2012.

[3] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," *Service Oriented Comput. Appl.*, vol. 5, no. 2, pp. 71–85, 2011.

[4] E. Avilés-López and J. A. García-Macías, "Tinysoa: A service-oriented architecture for wireless sensor networks," *Service Oriented Comput. Appl.*, vol. 3, no. 2, pp. 99–108, 2009.

[5] K. K. Khedo and R. Subramanian, "A service-oriented component-based middleware architecture for wireless sensor networks," *Int. J. Comput. Sci. Netw. Secur.*, vol. 9, no. 3, pp. 174–182, 2009.

[6] R. Kyusakov, J. Eliasson, J. Delsing, J. Van Deventer, and J. Gustafsson, "Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 43–51, Feb. 2013.

[7] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[8] M. Sonntag and D. Karastoyanova, "Next generation interactive scientific experimenting based on the workflow technology," in *Proc. MS10*, 2010, pp. 349–356.

[9] K. Lee, J. Jeon, W. Lee, S. Jeong, and S. Park, "Qos for web services: Requirements and possible approaches," *W3C Work. Group Note*, vol. 25, no. Nov., pp. 1–9, 2003.

[10] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Sci., Serv. Agents World Wide Web*, vol. 1, no. 3, pp. 281–308, 2004.

[11] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proc. 18th Int. Conf. World Wide Web*, 2009, pp. 881–890.

[12] J. El Hadad, M. Manouvrier, and M. Rukoz, "Tqos: Transactional and QoS-aware selection algorithm for automatic web service composition," *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 73–85, Jan.-Mar. 2010.

[13] T. Yu, Y. Zhang, and K. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Trans. Web*, vol. 1, no. 1, 2007, Art. no. 6.

[14] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 387–409, May/Jun. 2011.

[15] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "Qos-driven runtime adaptation of service oriented architectures," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2009, pp. 131–140.

[16] W. Heinzelman, "Application-specific protocol architectures for wireless networks," PhD Thesis, Dept. Electr. Eng. Comput. Sci., Massachusetts Institute Technology, Cambridge, MA, USA, 2000.

[17] L. Zeng, et al., "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[18] Y. Liu, A. H. Ngu, and L. Z. Zeng, "Qos computation and policing in dynamic web service selection," in *Proc. 13th Int. World Wide Web Conf. Alternate Track Papers Posters*, 2004, pp. 66–73.

[19] A. Afshari, M. Mojahed, and R. Yusuff, "Simple additive weighting approach to personnel selection problem," *Int. J. Innovation Manag. Technol.*, vol. 1, no. 5, pp. 511–515, 2010.

[20] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, Jun. 2007.

[21] Y. Qi and B. Athman, "Efficient service skyline computation for composite service selection," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 776–789, Apr. 2013.

[22] J. Li, X. Zheng, et al., "An efficient and reliable approach for quality-of-service aware service composition," *Inform. Sci.*, vol. 269, pp. 238–254, 2014.

[23] G. Canfora, et al., "A lightweight approach for QoS-aware service composition," in *Proc. 2nd Int. Conf. Service Oriented Comput.*, 2004, pp. 36–47.

[24] J. Ko, C. Kim, and I. Kwon, "Quality-of-service oriented web service composition algorithm and planning architecture," *J. Syst. Softw.*, vol. 81, no. 11, pp. 2079–2090, 2008.

[25] C. Jayapal and S. Vembu, "Adaptive service discovery protocol for mobile ad hoc networks," *Eur. J. Scientific Res.*, vol. 49, no. 1, pp. 6–17, 2011.

[26] Z. Zhang, W. Sun, W. Chen, and B. Peng, "An integrated approach to service selection in mobile ad hoc networks," in *Proc. 4th Int. Conf. Wireless Commun., Netw. Mobile Comput.*, 2008, pp. 1–4.

[27] C. Fok, G. Roman, and C. Lu, "Adaptive service provisioning for enhanced energy efficiency and flexibility in wireless sensor etworks," *Sci. Comput. Program.*, vol. 78, pp. 195–217, 2012.

[28] F. Delicato, F. Protti, L. Pirmez, and J. F. de Rezende, "An efficient heuristic for selecting active nodes in wireless sensor networks," *Comput. Netw.*, vol. 50, no. 18, pp. 3701–3720, 2006.

[29] V. Pilloni, M. Franceschelli, L. Atzori, and A. Giua, "A decentralized lifetime maximization algorithm for distributed applications in wireless sensor networks," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 1372–1377.

[30] W. Li, et al., "Efficient allocation of resources in multiple heterogeneous wireless sensor networks," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1775–1788, 2014.

[31] E. Tong, et al., "Bloom filter-based workflow management to enable qos guarantee in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 39, pp. 38–51, 2014.
[32] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[33] E. Al-Masri and Q. Mahmoud, "Investigating web services on the world wide web," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 795–804.
[34] Z. Zheng, Y. Zhang, and M. Lyu, "Distributed QoS evaluation for real-world web services," in *Proc. 8th Int. Conf. Web Serv.*, 2010, pp. 83–90.
[35] M. Berkelaar, J. Dirks, K. Eikland, and P. Notebaert, "Lpsolve: A mixed integer linear programming (milp) solver," 2007. [Online]. Available: http://sourceforge.net/projects/lpsolve

Endong Tong received the PhD degree from High Performance Network Laboratory, Chinese Academy of Sciences, Beijing, China, in 2013. He is currently an assistant professor of the Electronic Design Automation Center, Chinese Academy of Sciences, Beijing, China. His current research interests include workflow management systems, services computing and sensor networks.

Lan Chen received the PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2001. She is currently the director of the Electronic Design Automation Center, Chinese Academy of Sciences, Beijing, China. Her research activities have been in integrated circuit designs and computer architecture. Her current research interests include low-power design of very-large integrated circuits, interconnect of things chip design and the internet of things.

Huizi Li received the MS degree from China Foreign Affairs University, Beijing, China, in 2015. She is currently a research assistant of the Electronic Design Automation Center, Chinese Academy of Sciences, Beijing, China. She currently holds the position in the Center of Science and Education as graduate education manager. Her research interests include services computing and the internet of things.

Making Neighbors Quiet: An Approach to Detect Virtual Resource Contention

Joel Vallone, *Student Member, IEEE*, Robert Birke, *Senior Member, IEEE*, and Lydia Y. Chen, *Senior Member, IEEE*

Abstract—It is imperative for public cloud providers to guarantee performance targets for tenants' virtual machines (VMs) while respecting strict business confidentiality, e.g., having no information on applications nor their performance. A large body of related work addresses the challenges of detecting performance interferences by leveraging client's quality of service (QoS) metrics, e.g., latency, and additional profiling servers. In this paper, we take the perspective of the cloud provider and propose a general black-box approach that detects different resource contentions by throttling neighboring VMs. Specifically, we design a three-phase detection algorithm that includes: (i) an alarm phase to identify statistical outliers using control charts; (ii) a passive clustering phase to match the current sample to historical behaviors; and (iii) an active throttling phase to discern contentions from application phase changes via throttling. The algorithm is specifically designed for scenarios where multiple co-located VMs request detection analysis simultaneously. We implement and evaluate the proposed three-phase algorithm on four latency sensitive applications, i.e., Wikimedia and three benchmarks from Cloudsuite. Our extensive experimental results show that we can reach an average detection accuracy above 90 percent while limiting the performance degradation experienced by offender workloads to short learning phases.

Index Terms—Contention detection, cloud, QoS

1 Introduction

Virtualization is the key technology enabling the cloud computing paradigm on infrastructure, where a large amount of diverse resources are multiplexed together to cater to an ever increasing number of services. Cloud providers boost system productivity, as well as lower their operational costs, by hosting tenants via virtual machines (VMs) [1]. A commonly seen practice is to collocate applications with disparate performance and resource requirements, such as latency sensitive applications and batch applications. VMs have offered solutions to the problem of software heterogeneity, but not necessarily to the problem of performance isolation [2]. As a result, tenant's perceived Quality of Service (QoS), e.g., request throughput and latency, often degrades on clouds [3], [4] due to underlying resource contention, and substantial financial penalties may be incurred by both cloud providers and their tenants. It is of paramount importance for cloud providers to (pro) actively manage resource contention [5].

The challenges of preventing resource contentions in the cloud start right from the very first step - contention detection. To isolate tenants on the same infrastructure via a virtualization layer, providers have a bird's eye perspective on resource consumption of all VMs but lack high-level application performance metrics associated with the QoS perceived by tenant's. Providers are challenged to indirectly infer application performance from simple low-level resource metrics that preserve tenants' business confidentiality. Moreover, as the cloud enables a higher level of resource multiplexing across heterogeneous VMs, the system complexity and the volatility of workload dynamics drastically increase. Contention detection is thus required to be adaptive to fast application dynamics, including examples of load changes or internal application phases changes [6], [7], across collocated VMs.

There is a significant body of prior art [8], [9], [10], [11], [12], [13], [14], [15], [16] addressing the issues of resource contention and performance interferences in Cloud. Their main focus is to detect the QoS drop, either indirectly by inference from low level metrics [8], [12], [14] or directly by measuring the tenant's perceived QoS [9], [10], [15]. The former centers only on significant resource contention that can lead to obvious QoS drop. The latter implicitly requires application information that might intrude on the business confidentiality of tenants. One of the key ideas behind detecting contention is to create an isolated execution environment, e.g., creating a clone VM on a separate and dedicated server [8] or off-line profiling [12], [15], so that one can clearly differentiate between resource contention from inherent application dynamics. In summary, the related work may fall short in detecting a wide range of resource contention, some of which might only have a minor impact on QoS, without requiring additional tenant information and profiling servers.

In this paper, we aim to answer the following challenging research question - how can cloud providers detect a wide range of resource contentions of collocated VMs in an on-line

- *J. Vallone is with the Cloud & Computing Infrastructure, IBM Research Zurich, Rüschlikon 8803, Switzerland. E-mail: jva@zurich.ibm.com.*
- *R. Birke and L.Y. Chen are with the Cloud Server Technologies Group, IBM Research Lab Zurich, Rüschlikon 8803, Switzerland. E-mail: {yic, bir}@zurich.ibm.com.*

Manuscript received 27 Sept. 2016; revised 29 May 2017; accepted 30 May 2017. Date of publication 28 June 2017; date of current version 13 Oct. 2020. (Corresponding author: Lydia Chen.)
Digital Object Identifier no. 10.1109/TSC.2017.2720742

and black-box fashion? To such an end, we develop a new algorithm, which automates the processes of monitoring and the proposed three-phase detection framework: (i) **A**larming behavior change, (ii) passive **C**lustering, and (iii) active **T**hrottling diagnosis. Herein, the algorithm will be referred as *ACT* for short. Our main design idea is that we actively throttle neighboring VMs whenever a behavior change is confirmed after the second phase of *ACT*, to create an isolated execution environment at a low overhead. *ACT* monitors low-level resource metrics whose representative patterns are stored in a sliding window fashion. To on-line differentiate the application dynamics from contention, we employ a combination of statistical and unsupervised machine learning techniques, i.e., control chart, history look-up, and k-mean clustering, using data from regular monitoring and active throttling. We extensively evaluate the proposed *ACT* on a prototype cloud that collocates different combinations of latency sensitive applications, i.e., Wikimedia and CloudSuite, and batch-like applications, i.e., PARSEC and Cachebench.

Particularly, *ACT* executes on each physical server and monitors metrics related to the shared resources on VMs in discrete windows: CPU, cache, main memory, network and disk. *ACT* consists of three phases, each of which requires different computational complexity. First, an alarm phase (detect) identifies statistical outlier sample units in the victim's VM resource metrics based on control charts. Second, a passive diagnosis phase (remember) tries to match the outlier sample to historical behavior to minimize the detection impact. Third, an active learning phase (learn) throttles neighboring VMs to distinguish contentions from VM behavior changes. The computation overhead increases in phases. *ACT* is able to parallel execute the first (alarm) and second (cluster) phase on multiple VMs which are collocated on the same physical node, whereas the throttle phase can only be executed one at a time. Our evaluation results show an average detection accuracy above 90 percent when collocating Wikimedia with PARSEC and Cachebench. We also show that the direct throttling overhead on neighboring VMs can be as low as 15 percent of completion time increase and further amortized across time.

Our scientific contribution is on methodology as well as on experimental validation. We develop a novel and generic three-phase detection algorithm, which can accurately detect different resource contentions for collocated VMs by active throttling. *ACT* preserves clients' confidentiality and incurs low detection overhead. Our evaluation is on a realistic testbed executing representative workloads, i.e., a combination of interactive and batch applications.

The remainder of the paper is organized as follows. Section 2 presents a motivation example, followed by the system overview in Section 3. We detail the monitoring component in Section 4 and the three-phase detection algorithm in Section 5. In Section 6, we present our evaluation results, followed by a discussion on related work in Section 7. Section 9 concludes with a summary of our work.

2 Motivation

Here, we visually illustrate the challenges in detecting resource contention using a case study on the Wikimedia application [17].

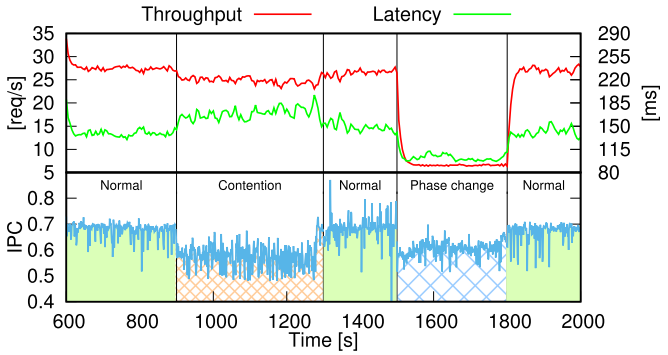


Fig. 1. Ambiguity problem: Just observing low level metrics does not help to differentiate contention from phase change.

When collocating VMs, the applications contend for shared physical resources which translates into changes in key metrics. One such metric is the instructions per cycle (IPC) used to measure program performance and contention [18], [19]. We deploy Wikimedia on a set of three VMs, each of which is hosted on a separate physical server. The detailed configuration of Wikimedia and servers can be found in Section 6. Table 1 illustrates the QoS metrics of the entire Wikimedia application together with the IPC measured on the Wikimedia frontend VM over time. We mark the entire observation with three labels, namely normal, contention and new phase. The contention period, from 900 to 1300 seconds, corresponds to the activation of a PARSEC benchmark (Blacksholes) [20] inside a VM collocated with the Wikimedia frontend VM. In addition, between 1500 and 1800 seconds, we drop the client request rate to emulate a phase change in the application dynamics.

As the IPC and throughput show similar decreasing patterns both during contention and new phase periods, it is not straightforward to differentiate contention caused by neighboring VMs from a new application phase: With a black-box approach, the challenge is to differentiate contention from application phase change only using hypervisor level metrics. To tackle such difficulty, one of the key elements in related work is to leverage the performance of the victim VM, i.e., Wikimedia, in an isolated execution, either by an initial profiling [12] or by cloning the VM on a dedicated hardware [8]. However, additional hardware and time are thus needed and this argues against the advantages of cloud - efficient and economic resource provisioning. Another observation worth noting is the non-linear relationship between the QoS metrics and contention. QoS may only show a clear difference when there is a high level of resource contention. For example, with the shown Wikimedia frontend, the difference of IPC in normal and contention period is roughly 25 percent, whereas the throughput and latency of the client requests show differences around 10 and 20 percent, respectively. As the CPU is rather saturated by the Wikimedia frontend VM and PARSEC VM, there exists a risk of drastic QoS drops when additional VMs appear. Low level resource metrics are more sensitive to potential contention from neighboring VMs, compared to high level QoS. One can view low-level resource contention as a necessary condition for interference. We thus advocate that detecting resource contention can be viewed as a conservative proxy for

TABLE 1
Monitored VM Metrics

Metric name	Source	Resource	Context
Instructions retired Per Cycle	`perf`	All	VM PIDs
CPU load buffer stalls cycles ratio	`perf`	Mem. & LLC	VM PIDs
LLC miss per K instructions	`perf`	LLC	VM PIDs
Net. card TX queue len. in Bytes	`tc`	Net. card TX	Virt. iface name
Disk read queue avg. wait time	`iostat`	Disk	Virt. disk name
Disk write queue avg. wait time	`iostat`	Disk	Virt. disk name

detecting QoS drops, without leveraging any application information. The immediate challenges of detecting contention arise from the large number of resource metrics, and the distributed nature of applications in the cloud.

3 System Overview

The contention detection system depicted in Fig. 2 operates in parallel to the tenants' VMs inside each hypervisor and is made up of two main component types: monitors, one for each running VM, and detectors, one for each hypervisor. Each monitor periodically collects the resource contention metric data of its coupled VM and feeds it to the detector residing on the same hypervisor. The detector can execute our three-phase detection algorithm for multiple VMs of interest, such as latency-sensitive application VMs. Each phase requires different degrees of computational complexity and results in different levels of detection accuracy. During the first phase (**A**larm), each detector processes the incoming monitoring stream to identify outliers signaling the insurgence of a possible contention. If outlier sample units are found, the detector goes into the second phase (**C**lustering). This passive diagnosis phase tries to infer contention by clustering the current contention sample to known VM behaviors. If the sample can not be classified using historical data, the active diagnosis phase (**T**hrottling) starts. During this phase the detector tries to remove any contention by throttling all neighbouring VMs to emulate an in-isolation run of the victim VM. If the sample before and during throttling can be divided into two distinct groups, the detector can classify the behavior as contention.

On the one hand, scale-out scalability is not an issue since the detection algorithm runs independently in parallel on all hypervisors. On the other hand, the throttle phase limits the number of active diagnosis to one at a time affecting scale-in scalability. In practice, this limitation is eased by the fact that the active throttle phase is necessary only in a few cases.

3.1 Assumptions

The detection algorithm relies on a set of underlying assumptions to work well. The main one is that the victim VM is in a steady state during each phase of the diagnosis process. By steady state we mean constant client load or computation load. In practice, load-balanced client-facing services and big batch workloads appear to be in steady

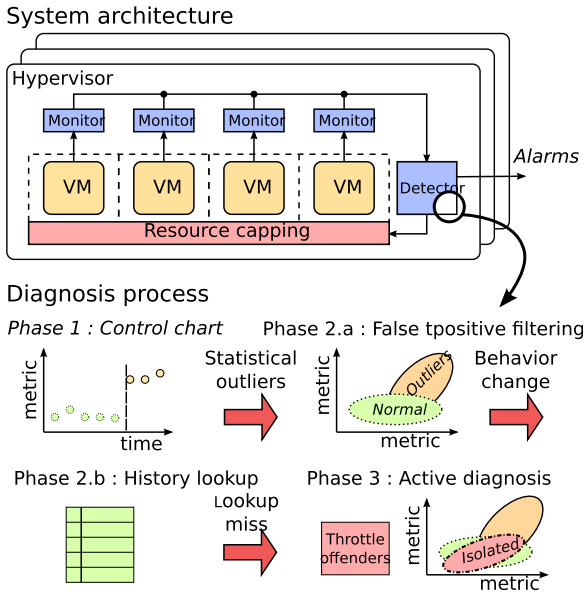


Fig. 2. System architecture: Each VM resource usage is monitored and fed into a three-phase detection algorithm, which relies on throttling VMs to discern the ground truth.

state when the observation window lasts a few tens of seconds. Another assumption is that neighbouring VMs can tolerate sporadic CPU capping. This is true for VMs running batch jobs but might be an issue for latency sensitive applications. Finally, the algorithm assumes to know an initial ground truth on the normal VM behavior. For the moment, we rely on normal samples gathered in isolation conditions. Lifting this last assumption is left for future work.

4 Monitor

We rely on a monitoring system which consists of many small monitor blocks, one for each running VM. Each VM is monitored directly from the hypervisor level making the whole monitoring infrastructure transparent to the VMs themselves. Each monitor collects different metrics using standard monitoring tools such as `perf`, `tc` and `iostat`. These tools are contextualized to monitor only the specific VM attached to the monitor. This is achieved by monitoring either a specific virtual device attached to the VM or filtering the thread group of the QEMU emulation process. The main focus is on contention metrics used by *ACT*. Each monitor is also able to collect load metrics. Table 1 summarizes the metrics used by the diagnosis tool, the associated monitoring tools and their contextualization.

As seen in Table 1, the focus is on metrics which allow contention on shared resources to be highlighted, since contention metrics are more informative on possible interference cases. To minimize measurement error, we limit the `perf` metrics so as not to exceed the number of multi-purpose hardware performance counters available on Intel[1] i7 CPUs.

The main requirement of the monitor block is to induce the least amount of overhead as possible. Each monitor can

1. Intel is a trademark or regsitered trademark of Intel Corporation or its subsidiaries in the US or other countries. Linux is a registered trademrk of Linus Torvalds in the US or other countries or both. Other product and service names may be trademarks or service marks of IBM or other companies.

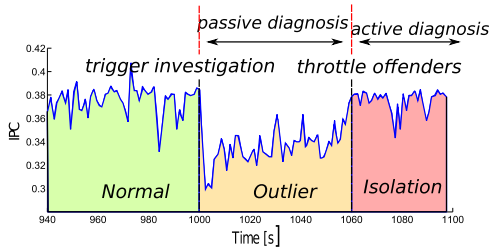


Fig. 3. Three-phase diagnosis cycle. Sample units are separated in three types of intervals: Normal, outlier, and isolation, corresponding to alarm, cluster, and throttle phase.

easily collect more metrics than the ones listed, including load metrics, but each additional metric increases the measurement overhead. Hence we limit ourself to this list.

Particular care has to be given to the set of metrics collected via `perf`, since `perf` relies on hardware performance counters. If the number of metrics exceed the number of performance counters available within the processor, `perf` time multiplexes the different metrics at the expense of measurement accuracy. Each processor in our testbed is equipped with four performance counters. Thus, we limit the monitored metrics to IPC, Last Level Cache (LLC) misses per K instructions and load buffer stalls. IPC is a good overall measure of CPU stall produced by contention. LLC misses highlight cache conflicts and load buffer stalls increase when the memory system is under heavy contention. We investigated the possibility to monitor other metrics such as L2 D/I misses or arithmetic unit stalls but did not retain them due to the low detection accuracy improvement over the retained metrics.

Limiting the number of low-level metrics collected by `perf` to the number of available hardware performance counters allows a new sample unit to be produced every second without loss of accuracy. Moreover, from preliminary experiments the 1-second sampling frequency results in a good compromise between accuracy, speed and monitoring overhead. All metrics are gathered at the same time and stored in a vector where each element is the measured value of each metric. This vector can be seen as a point in a multidimensional space with one dimension per metric.

5 Detector

The detector block collects the measurement data from all VM monitors collocated under the same hypervisor and runs the three-phase on-line detection algorithm. The detection algorithm can run in parallel on the monitored VMs during the behaviour change detection and the passive diagnosis but only one VM can be actively diagnosed at a time because of neighbouring VMs throttling. The algorithm aims to detect VMs contending on shared resource(s). Contention relates to a bad resource sharing which can mainly be observed as a step change on related contention metric(s).

As an example we present a simplified case where we consider only one metric. Fig. 3 depicts the IPC of a victim VM across time. At the beginning, the server is in contention free behavior (normal). We consider the sample as normal when its values match isolation conditions. At 1000 seconds, we have a behavior change in the server due to a neighbouring VM starting a heavy CPU-intensive task (outlier). This creates a change in the IPC trace. The first phase of the algorithm continuously monitors the VM to identify such a change between the normal and outlier parts across any of the metrics. Once in the outlier part, the second phase of the algorithm tries to classify the new behavior by collecting outliers and comparing them to historical data. In the case of miss, the algorithm goes into the third and last phase: throttling of the neighbouring VM(s) (isolation). The three-phase detection loop is summarized in Algorithm 1, whereas each phase is described in detail in the following sections.

The challenge is to differentiate contention from application phase change. An application phase change can be observed when the running code base inside the VM changes creating different measurement patterns due to the program entering a different part of its code. For latency sensitive workloads, the variation of client load is also considered as a phase change. To find the ground truth, we isolate the VM from its neighbors by throttling. If it is an application phase change, the measurements are not affected by the neighbouring VMs and hence should remain unchanged even in isolation. If it is contention, the measurements should improve when in isolation. In Section 6 we show that the throttling cost, even in terms of performance drops of the neighbouring VMs, is limited.

Algorithm 1. *ACT*– Detector Loop

```
while true do
  x = monitorVM()
  updateSample(x, W_normal)
  if detectChange(W_normal) == false then          ▷ Phase 1: Alarm
    continue
  end if
  for i = 0; i < w; i++ do                         ▷ Phase 2: Cluster
    x = monitorVM()
    updateSample(x, W_outlier)
  end for
  o = filterFalsePositives(W_normal, W_outlier)    ▷ Phase 2.a
  if o == falseAlarm then
    continue
  else if historyLookup(W_outlier) == match then   ▷ Phase 2.b
    signal outcome of match
  else
    throttleNeighbourVMs()                         ▷ Phase 3: Throttle
    for i = 0; i < w; i++ do
      x = monitorVM()
      updateSample(x, W_isolation)
    end for
    unthrottleNeighbourVMs()
    if classifyBehavior(W_normal, W_outlier, W_isolation) != failure then
      signal outcome of classification
    end if
  end if
end while
```

5.1 Alarm Phase (I): Behavior Change Detection

The goal of the first phase is to have a computational cheap way of detecting behavior change to minimize the detection overhead. In this phase, fast detection is preferred over accuracy, since the accuracy will be achieved by the subsequent phases.

During this phase, the detector continuously monitors all the metrics of all the VMs independently. We rely on

Shewhart control chart [21] to detect operation anomalies on any of the metrics. The basic idea is that during an anomaly the measures will start deviating and the anomaly can be detected by setting a threshold on the deviation. Shewhart control chart is known as a robust statistical quality control which does not make any assumption on the underlying probability distribution. The method uses a moving window $\mathbf{W} = \{x_{t-1}, x_{t-2}, \ldots, x_{t-w}\}$ of size w containing the last sample units to compute the moving sample mean $\overline{x}$ and range mean $\overline{MR}$ as

$$\overline{x} = \frac{1}{w}\sum_{i=1}^{w} x_i \tag{1}$$

$$\overline{MR} = \frac{1}{w-1}\sum_{i=2}^{w} |x_i - x_{i-1}|. \tag{2}$$

When a new measurement x_t arrives, an anomaly is signaled if condition (3) is not satisfied

$$\overline{x} - 2.66\,\overline{MR} \leq x_t \leq \overline{x} + 2.66\,\overline{MR}, \tag{3}$$

where 2.66 is the commonly recommended value given that the moving range is estimated using two points [21]. In practice this method works but is too sensitive to noise. To make the detection resilient to noise, we count the number n of consecutive outliers where sample units are labelled as outliers based on the estimated mean $\overline{x}$, standard deviation σ and the three-sigma rule, i.e., a new value x_t is an outlier if condition (4) is not satisfied

$$\overline{x} - 3\,\sigma \leq x_t \leq \overline{x} + 3\,\sigma, \tag{4}$$

where

$$\overline{x} = \frac{1}{w}\sum_{i=1}^{w} x_i,\ \ \sigma = \sqrt{\frac{1}{w-1}\sum_{i=1}^{w}(\overline{x} - x_i)^2}.$$

Since we monitor sets of metrics the above condition is applied separately to each metric and the first to violate the threshold raises the alarm.

Outlier sample units are counted but not immediately added to the moving window $\mathbf{W}$ of size w containing the last sample units $\mathbf{W} = \{x_{t-1}, x_{t-2}, \ldots, x_{t-w}\}$. If n is greater than a threshold n_t, the next phase starts, otherwise the outliers are added to $\mathbf{W}$. In the following, we set $n_t = 5$ and $|W| = 30$ which proved to be good choices in preliminary experiments.

5.2 Clustering Phase (II): Passive Diagnosis

The goal of the second phase is twofold. The first goal is to classify the outlier sample in the alarm phase as noise or representative of a new system behavior, i.e., false positive filtering. The second goal is to discern contention from phase change based on historical data, i.e., history lookup, to avoid the cost of an active diagnosis (throttling phase). Please note that while phase 1 considers each metric independently, from here on we consider the whole VM measurement vector as a representation of a point in the multidimensional metric space.

False Positive Filtering. The problem with phase 1 is that it can easily capture noise and result in false alarms as observed from preliminary experiments. To reduce the number of false positives, phase 2 first performs a filtering based on clustering. The algorithm accumulates measures for the outlier sample $\mathbf{W_{outlier}}$ from the new system behavior identified by phase 1 having the same size w as the normal sample $\mathbf{W_{normal}}$ collected in the previous behavior. After that, it runs K-means on the union $\mathbf{W_{normal}} \cup \mathbf{W_{outlier}}$ with $K = 2$. The rationale is that if phase 1 correctly detected a new system behavior, clustering should highlight two very distinct samples of roughly equal size (since $|\mathbf{W_{outlier}}| = |\mathbf{W_{normal}}|$): one representing the previous normal behavior mostly composed of normal sample units and one representing the new behavior mostly composed of outlier sample units. On the contrary, if phase 1 triggered on noise, this clear separation will not be true.

Algorithm 2 details all the filtering steps. The *leader()* function, given as input a set of points and a threshold C, returns the identifier of the cluster which is both the biggest cluster in terms of number of points and comprises at least C percent of the total points. If no such cluster is found, a negative value is returned. This ensures that the normal sample clearly defines one cluster while the outlier sample clearly defines the other cluster. More precisely, we never want a negative return value from *leader()* and the return values on the two sets should not be equal. The C parameter can be considered as a characterization threshold over the clustering degree of a subset of points. In practice, we want a high characterization on the normal cluster, hence $C_{normal} = 70\%$, while we want enough outlier sample units in the new cluster to highlight a tendency rather than noise. We investigate the meaning of enough in Section 6.3 by varying $C_{outlier}$.

Algorithm 2. False Positive Filtering

```
P = W_normal ∪ W_outlier
clusters = kMeans(P, 2)
n = clusters.leader(W_normal, C_normal)
o = clusters.leader(W_outlier, C_outlier)
if n = −1 ∨ o = −1 then
  return falseAlarm
else if n = o then
  return falseAlarm
else
  continue diagnosis
end if
```

History Lookup. For each VM the past known normal and contention behaviors are stored in a history table populated by phase 3. For storage compactness and computational speed, we approximate each cluster as a sphere in a normalized multidimensional space of the metrics. This allows us to store each cluster k simply by its outcome, i.e., contention or phase change, its centroid $\mathbf{C_k}$ and its radius r_k, together with normalization factors, i.e., mean $\mu_{i,k}$ and standard deviation $\sigma_{i,k}$, for each metric i. At this point a history lookup of a new VM behavior simply translates into checking if a majority of outlier sample units are within a cluster radius from the VM history table based on the euclidean distance between the cluster centroid and each of the normalized sample units.

More formally, we first normalize each vector element x_i of sample unit vector $\mathbf{x}$ using $\mu_{i,k}$ and $\sigma_{i,k}$ of the cluster k to obtain a normalized sample unit $\mathbf{x_k^n}$

$$x_i^n = \frac{x_i - \mu_{i,k}}{\sigma_{i,k}}. \quad (5)$$

Once normalized, $\mathbf{x_k^n}$ is part of cluster k if

$$||\mathbf{x_k^n} - \mathbf{C_k}|| < r_k, \quad (6)$$

under euclidean distance. μ_i, σ_i and r are computed from the cluster sample units during phase 3. While μ_i and σ_i are trivially the mean and standard deviation over those values, to avoid ambiguous points the radius r is computed as the 90th percentile of the euclidean distance distribution of the sample units. Note that normalization is necessary because each metric has its own range of values and thus a direct comparison across different dimensions would be difficult. In practice, we require at least $C_{history}$ percent of outlier sample units to be part of a cluster to signal a lookup hit, where $C_{history}$ is set to 70 percent. This operation is repeated over all history table entries. In the case of a hit, the diagnosis is stopped and the outcome stored in the history table is returned. In the case of multiple hits, the cluster with the highest percent of matching sample units is selected. Finally, in the case of a miss, the active diagnosis phase is invoked.

Over time the size of the history table grows as the detector identifies new VM behaviors. To avoid excessive slowdowns and memory footprints, one can easily limit the number of entries in the history table, but at the cost of extra active diagnosis phases. In practice, this problem is limited due to the fact that only a few new behaviors are discovered, mainly at system startup, while later the algorithm relies on history hits as seen in Section 6. The trade off between the table size and the impact of extra throttling phases is left as interesting future work.

5.3 Throttling Phase (III): Active Diagnosis

The goal of the third phase is to isolate the victim VM from all neighbouring VMs to identify the ground truth. Contrary to the related work, to isolate the VM we rely on throttling which has several advantages. (i) It is easy to deploy: it does not rely on any special feature not readily available in most modern hypervisors. (ii) It is completely application transparent: it does not require any prior knowledge on or cooperation from the VMs. (iii) Throttling by itself bears a minimal cost, especially compared to more complex schemes such as priori off-line VM characterization or VM cloning. The main disadvantage is the performance impact on the applications running inside the throttled VMs. However, in Section 6, we show that the impact, although perceivable, can be maintained within limits both time-wise and performance degradation-wise. Moreover, after some initial learning phases to populate the history table, throttling is mostly avoided.

In more detail, if phase 2 is not able to classify the outliers, phase 3 throttles all the neighbouring VMs capping their CPU usage and accumulating a new isolation sample $\mathbf{W_{isolation}}$ of same size w. After that, the same principles used in phase 2 are applied to discern phase change from contention. Algorithm 3 details the steps. We start with K-means using $K = 2$ on the union $\mathbf{W_{normal}} \cup \mathbf{W_{outlier}} \cup \mathbf{W_{isolation}}$, and identify the main cluster in all three samples via the *leader()* function. For an accurate diagnosis, we want a good characterization on the isolation sample by setting $C_{isolated} = 70\%$ to be noise tolerant but keep the clusters sufficiently reliable.

Algorithm 3. Behavior Change Classification

$\mathbf{P} = \mathbf{W_{normal}} \cup \mathbf{W_{outlier}} \cup \mathbf{W_{isolation}}$
clusters = kMeans($\mathbf{P}, 2$)
n = clusters.leader($\mathbf{W_{normal}}, C_{normal}$)
o = clusters.leader($\mathbf{W_{outlier}}, C_{outlier}$)
i = clusters.leader($\mathbf{W_{isolation}}, C_{isolated}$)
if $n = -1 \vee o = -1 \vee t = -1$ **then**
  return diagnosisFailure
else if $i = n$ **then**
  return contention
else if $i = o$ **then**
  return newPhase
end if
return diagnosisFailure

If the cluster of isolation sample units matches the cluster of normal sample units, the diagnosis classifies the outliers as contention. The rationale is that, by removing/mitigating the impact of the neighbouring VMs, the victim VM behaves as normal because the behavior change was contention induced. On the contrary, if the cluster of isolation sample units matches the cluster of outlier sample units, then we witness a new VM behavior because the neighbouring VMs had no influence on the behavior change of the victim VM. In either case, the diagnosis succeeded and the history table is updated with a new entry. In the case of a contention, this new cluster is based on the outlier sample, whereas in the case of a new phase, the cluster is based on the isolation sample. If neither condition is true, the diagnosis failed. Independently of the outcome, the detection algorithm restarts from phase 1 with its state reinitialized using the last normal sample in the case of diagnosis failure or contention, or using the outlier sample in the case of phase change.

5.4 Parallel Analysis

We illustrate how to extend *ACT* to the scenario of multiple VMs of interest and conduct a so-called parallel analysis. In contrast to the previous section, *ACT* monitors the performance metrics of, and detects contention across multiple VMs concurrently. This introduces two additional complexities to be addressed by the detector. First, while the alarm and cluster phases are completely passive and can easily be performed in parallel on each VM under scrutiny, the throttle phase is active and requires coordination across all the VMs. Second, VMs might switch between active and inactive states during detection which can introduce extra perturbations to the detector. In the following, we first illustrate an example of applying *ACT* on three latency-sensitive VMs hosted on the same physical server and then explain how to coordinate throttling and handle VM state switches in practise.

3-VM Example. We apply *ACT* on the three VMs of interest shown in Fig. 4. VM 1 and VM 2 are active during the whole observation period, whereas VM 3 becomes activate at time t_a. We note that there might be other VMs collocated on the same physical server which do not enter in the set of VMs of interest, e.g., VMs running latency-insensitive batch workloads. In our example VM 1 and VM 2 start in a condition classified as normal, until there is a change in the server

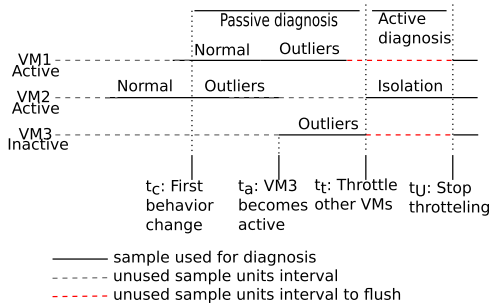


Fig. 4. An illustration of *ACT* parallel analysis on three VMs of interest. The alarm and cluster phases are executed independently for each VM, whereas the active throttle phase is only allowed for VM 2.

load due to any of the hosted VMs. This change is first detected by the alarm phase of VM 2 followed by VM 1 which independently switch to collecting outliers. Upon collecting enough outliers for the passive diagnosis of the cluster phase, *ACT* decides if an active throttle phase is needed for individual VMs. In this example, *ACT* reaches the decision of entering the throttle phase for VM 2 and throttles all its neighbouring VMs on the same server for the purpose of collecting the isolation sample.

Coordination of Throttle Phase. While the passive diagnosis of the cluster phase of *ACT* can be operated in parallel on all VMs of interest, *ACT* allows only one VM of interest to enter the throttle phase. In other words, *ACT* waits for the cluster phase for all VMs to finish before admitting one VM to the throttle phase. When only one VM is investigated, the passive diagnosis lasts for the duration of collecting the VM outlier sample. However, when multiple VMs are monitored, the passive diagnosis may last longer. Indeed, when a VM experiences a behaviour change on shared resources, it is likely that the other VMs are affected too, see VM 2 and VM 1 in our example. Thus, the cluster phase accepts other VMs for investigation until all have both their normal and outlier samples available, i.e., t_t in our example. Then, based on the outcome of the diagnosis of the cluster phase, if multiple VMs need to execute the active diagnosis, only one is selected, and the others fall out the diagnosis iteration. We prioritize the VM having the least active diagnosis count, i.e., the smallest history.

Handling VM State Switches. We consider the VM activity state when running the diagnosis. We define a sample unit as active if at least one of the measured resources (i.e., CPU, disk or network) has a utilization above a given threshold, e.g., 1 percent in our experiments. During each diagnosis, samples must have a majority (90 percent) of their sample units marked as active. Otherwise, the diagnosis fails. Similarly we define a VM as inactive after twenty consecutive VM sample units are marked as inactive, and as active otherwise. Once a VM becomes inactive we exclude this data from the diagnosis. In contrast, a newly active VM may already start in a contention scenario (see VM 3 in our example). Hence the newly-active sample is considered as an outlier sample in the cluster phase. If the sample matches a history entry marked as *phase-change*, the sample is considered as normal and the diagnosis finishes. Otherwise, we need to collect an initial (ab)normal sample. We thus need to enter active diagnosis and throttle neighbors for such a VM so as to figure out if it starts with a normal sample.

6 Evaluation

In this section, we present an extensive evaluation of *ACT* on a prototype system that collocates latency sensitive and batch-like applications. We show the effectiveness of *ACT* in detecting resource contention for latency-sensitive applications under various collocation scenarios and parameter settings. The specific metrics of interest are detection accuracy, delay, and overhead that is measured on the throughput degradation of batch-like applications. We first show detection rates and delay of scenarios where one latency-sensitive is collocated with one batch-like application. We then extend to parallel scenarios where multiple *ACTs* monitor latency-sensitive applications in parallel. Moreover, via long-running time-varying experiments, we show that *ACT* can effectively differentiate contention, from internal program change as well as workload dynamics.

6.1 Testbed Setup

The testbed is composed of four identical physical servers connected via a star topology to a Gigabit switch. Each server runs Ubuntu server 14.04 LTS and is equipped with 16 GB of DDR3 RAM, a 4-core Intel[1] Core i7-3820 processor @ 3.6 GHz with SMT, one 2-TiB Sata III 7200 rpm hard disk, and one Gigabit Ethernet adapter. Three servers host the VMs used to deploy the applications. We run QEMU v2.0 with KVM on top of Linux[1] kernel 3.13 as the hypervisor. Each VM comprises two virtual CPUs and 2 GB of RAM. In most cases, we host two VMs on each server: (i) one VM running latency-sensitive application or its component, and (ii) the other VM running batch workloads to generate contention. The fourth server is used as the experiment orchestrator and load generator.

Latency-Sensitive Applications. We use Wikimedia as a representative cloud workload [17]. Wikimedia is a latency-sensitive three-tier web application composed of Apache (v2.4.7) plus PHP (v5.5.9) as the application server frontend, Memcached (v1.4.14) as the in-memory key-value store and MySQL (v5.5.40) as the database backend. Each component is deployed into separate VMs which in turn are hosted on separate servers. As a reference, the maximum sustainable throughput of this setup is 38 request/s.

We also use three of the Cloudsuite [22] latency-sensitive workloads to further validate the effectiveness of *ACT*. First, there is data-serving which runs Cassandra (v0.7.3), a distributed scale-out database. The second workload is data-caching, a Memcached (v1.4.14) instance storing a twitter data-set. The last workload, Media-Streaming, is a Darwin Streaming server (v6.0.3) stressed by Faban RTSP clients.

Emulated Contention. To create contention, we spawn, in parallel with Wikimedia and/or Cloudsuite workloads, neighboring VMs running two different resource intensive batch workloads resulting in two different types of contention. On the one hand we use a subset of scientific applications from PARSEC 3.0 [20], i.e., Fluidanimate, Blackscholes and Freqmine, characterized by regular but intensive CPU and memory usages. We run these benchmarks using the native input dataset size. On the other hand we use Cachebench [23] which gradually pollutes the whole last-level cache producing extremely noisy usage traces spanning from low CPU and cache usage to full CPU and cache usage. Since Cachebench has a very short runtime, we

TABLE 2
Extensive Contention Analysis Measured via Percentage of Degradation on Instructions per Cycle (IPC), LLC Misses per Instruction (LMI) and Disk Write Wait Time (DWW): Comparing Before and After Being Collocated with Freqminie, Fluidanimate, Blackscholes, and Cachebench

Victim / Contender	Darwin			Memcached			Cassandra			Wikimedia		
	IPC	LMI	DWW	IPC	LMI	DWW	IPC	LMI	DWW	IPC	LMI	DWW
Freqmine	10.1	58.4	77.7	21.0	29.2	0	9.7	106.8	87.5	9.3	16.1	42,361.2
Fluidanimate	10.5	166.8	8.4	21.4	110.8	0	9.7	161.6	-13.2	13.5	17.4	6,815.3
Blackscholes	9.8	85.3	28,814.9	20.4	46.9	0	-14.6	74.6	76.7	15.6	27.5	76.8
Cachebench	10.2	166.3	105.0	20.5	114.6	0	-17.6	147.0	-5.8	11.7	36.3	17.4

create longer contention periods running it in an infinite loop and killing the loop after a timeout. Throttling is based on the `cgroup` kernel module which allows the CPU usage bounds to be specified based on process identifiers (PIDs). Since each Wikimedia component is placed on a different server and has different resource usages, this setup allows multiple contention mixes to be studied at the same time.

6.2 Contention Analysis

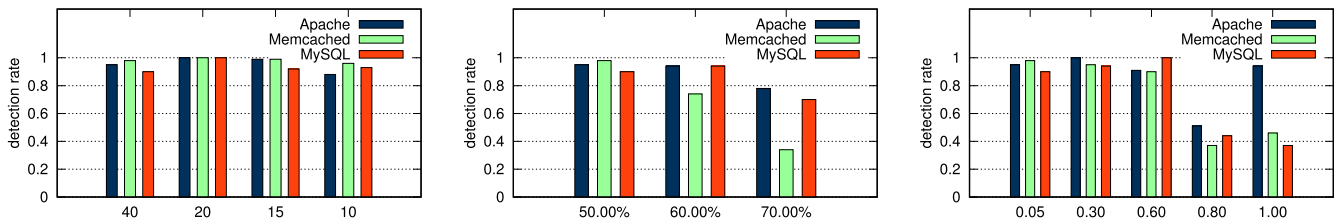
Prior to presenting *ACT* results, we first conduct a detailed contention analysis that provides the ground truth and root cause on how latency-sensitive applications and batch workload contend for different resources. Here, we consider all 16 scenarios of combining four latency-sensitive applications and four batch workloads on a single server. We look into three types of resources: CPU, memory and disk. We analyze the contention patterns first from the perspective of latency-sensitive applications and then of the batch workloads.

Setup and Metrics. Specifically, we focus on three metrics: IPC, last-level cache misses per instruction (LMI), and disk write wait time (DWW) which are indicators for contention on three different resource types, i.e., CPU, memory, and disk, respectively. To quantify the contention effect from the neighboring batch workload, we use the degradation of metrics, comparing its values before and after having the neighboring batch workload. We thus use offline profiling to compare these metrics collected on VMs running a latency-sensitive application in isolation against the collocated scenario with a neighboring VM running a batch workload. In addition to Wikimedia, we use three latency-sensitive Cloudsuite benchmarks: media streaming (Darwin), data serving (Cassandra) and data caching (Memcached). As for the batch workloads, we use Blackscholes, Freqmine and Fluidanimate from PARSEC and Cachebench [23].

From Latency-Sensitive Applications. Table 2 summarizes the results for media streaming, data serving, data caching and web serving, respectively. Each figure shows the metrics degradation/improvement in percentage after being collocated with batch workloads. The degradation is computed as the performance difference divided by the original performance value. For the IPC, we compute the difference by the value before collocation subtracted by the value after collocation, whereas for LMI and DWW we compute the difference by the value after collocation subtracted by the value before collocation. This is to reflect that a higher IPC value is deemed a better performance but higher LMI and DWW values are deemed inferior performance. Negative values of degradation mean that there is actually a performance improvement after collocating with neighboring VMs.

From the results one can observe that VMs do interfere with each other and the degree/type of interference depends on specific combinations of applications and benchmarks. Shown in Table 2, Memcached has the highest IPC degradation among all four applications. Though Memcached can effectively use the memory space to boost the data store, it is also known to have scaling issues when increasing the number of threads [24], indicating that both CPU and memory can be resource bottlenecks. As for Wikimedia, it is sensitive to disk activities, a counter-intuitive finding. Indeed, Wikimedia frontend consists of Apache passing requests to a stateless middleware script written in PHP without explicitly using disk. However, the default configuration of Apache used in our experiments records all requests to a log file on disk. Table 2 shows that Cassandra's DWW is less affected because the workload used to stress the service is read heavy. Darwin is the most memory sensitive application. This is due to the fact that it uses in-memory caching of the content and that the test videos are of comparable size to the last-level cache.

From Batch Workloads. Freqmine, Fluidanimate and Blackscholes represent compute-intensive real-world applications and at the same time Freqmine and Fluidanimate (particularly) are also memory-intensive [20]. Moreover all three use the disk to either output their results (Fluidanimate and Blackscholes) or load the input datasets (Freqmine). As for Cachebench, it is a benchmark explicitly written to stress the cache, making it both memory- and compute-intensive. These resource characteristics are clearly visible across all four figures. Let us first focus on the IPC degradation which is rather similar in call cases. With a closed check one can see that Fluidanimate and Cachebench typically cause higher IPC degradation, compared to Freqmine and Blackscholes. This observation demonstrates how all four batch workloads compete for CPU with the latency-sensitive applications. In terms of memory contention, Fluidanimate and Chachebench create the highest LMI degradations as expected. It is worth noting that sometimes the collocation of VMs can bring seemingly beneficial effects. This is the case in Table 2 which shows an improvement of the IPC and DWW for Cassandra, i.e., the negative degradation numbers. We attribute this positive effect to the CPU power governor which increases the clock frequency in response to an overall higher load.



(a) Sample size $|\mathbf{W_o}.| = |\mathbf{W_i}.|$ [#] (b) $C_{outlier}$ [%] (c) CPU limit Th_{CPU} [cores]

Fig. 5. Contention detection rate under different parameter settings of *ACT*.

6.3 Effectiveness of *ACT* on Wikimedia

Here, we show a sensitivity analysis on the impact of the *ACT* parameters on detecting resource contention caused by the collocated batch applications for Wikimedia. First, the sample sizes of $\mathbf{W_{outlier}}$ and $\mathbf{W_{isolation}}$ used in the passive and active diagnosis phase, respectively. Intuitively, the bigger the sample, the more time is required to collect it but the lower the risk of considering noise as an actual trend. Note that, on average, one sample unit is produced every 1.15 seconds. Second, the CPU capping limit Th_{CPU} imposed during throttling. The higher the capping the less impact we have on the neighboring VMs. Of course, with less capping the sample of the victim VM will not represent a perfect isolation case and the diagnosis is more likely to fail. Third, the threshold $C_{outlier}$ used to find the outlier cluster. The higher this threshold, the clearer the trend must be to generate alarms, lowering the risk of false positives. However, *ACT* might miss contention cases because the threshold is too restrictive. While tuning one parameter we use the following conservative values for the other two: $|\mathbf{W_{outlier}}| = |\mathbf{W_{isolation}}| = 40$ sample units, $Th_{CPU} = 0.05$ cores and $C_{outlier} = 50\%$.

Experiments consist of Wikimedia running at full load, while we randomly spawn on each server an instance of either PARSEC or Cachebench inside the neighboring VM. 2-minute breaks are spaced out by offender intervals lasting between 3 to 4 minutes. There is an independent 50 percent probability that the next interval will spawn a batch job on each server. Whenever a batch job is spawned, contention systematically occurs. Each run lasts 3 hours and is repeated 3 times. In the following we present the average results across the runs.

6.3.1 Detection Rate

We first evaluate the detection rate defined as the ratio of correctly identified contention intervals (true positives) over all known contention intervals. We consider a contention interval to be correctly identified if one or more alarms were raised in it. An ideal algorithm always correctly identifies contention intervals, i.e., detection rate equal to 1. Fig. 5 shows how the detection rate changes under different tunable parameter values.

We start by varying the number of sample units $|\mathbf{W_{outlier}}|$ and $|\mathbf{W_{isolation}}|$ per diagnosis interval (see Fig. 5a). The best results, detection rates equal or close to 1 across all three Wikimedia components, are obtained with 15-20 sample units. With less units the measurement noise is higher and the diagnosis is more likely to fail. With more units the noise effects are reduced, but since each diagnosis takes longer, less diagnosis cycles can be repeated in the case of diagnosis failures before the contention disappears. Naturally this case is directly related to the expected contention length. Both conditions result in lower detection rates, especially for the Apache and MySQL components.

Moving on to the impact of the outlier threshold $C_{outlier}$, one can observe in Fig. 5b a clear decreasing trend: the higher the threshold, the lower the detection rate. This is true for all three Wikimedia components. Indeed, when we require a clearer clustering, only the VMs under heavy or obvious contention will maintain a high detection rate. Memcached is especially affected with significant drops at both 60 and 70 percent. This is because this VM has the least resource usage and thus its sensitivity to neighboring VMs contending for shared resources is the lowest.

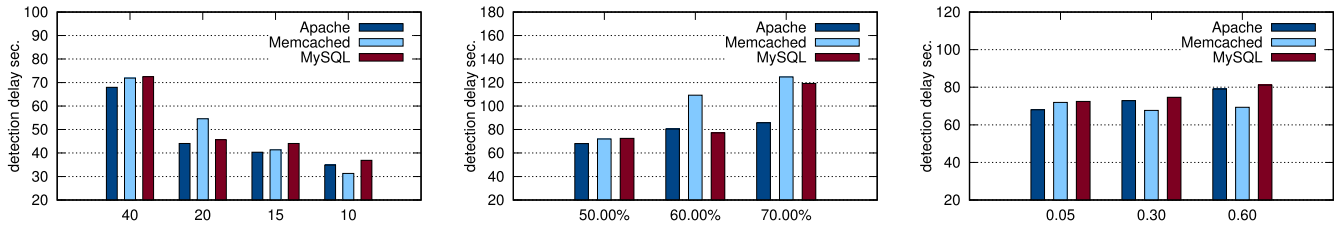
Comparing the detection rate against the CPU capping limit Th_{CPU}, we observe that the active diagnosis phase does not need to fully throttle the neighboring VMs to be able to identify contentions. As highlighted in Fig. 5c, the detection rate remains stable until Th_{CPU} reaches 0.6 cores. Beyond this, it rapidly degrades. However, even if we show the average across both contention scenarios, PARSEC and Cachebench have different influences. Apache is more sensitive to the aggressive CPU usage of PARSEC, while the other two Wikimedia components are more sensitive to Cachebench. Using the more aggressive neighboring VM, contention can be discerned even in low isolation conditions, otherwise the detection collapses. The fact that even low CPU throttling is sufficient to identify contention is desirable because it directly reduces the performance impairment experienced by the throttled VMs.

Parameter Tuning. Overall, the best detection rates are obtained using $15 < |\mathbf{W_{outlier}}| = |\mathbf{W_{isolation}}| < 20$ for the best tradeoff between measurement noise and diagnosis cycle rate. The CPU cap can be rather high as long as $Th_{CPU} < 0.6$ cores. Finally the cluster threshold should not be too strict with $C_{outlier} < 60\%$.

6.3.2 Detection Delay

A detection system should not only be accurate, but also fast. We define the contention detection delay as the time elapsed between the start of a contention interval and the first alarm raised by the detector. Fig. 6 shows the average detection delay over all correctly identified contention intervals across the same previously presented parameter sweeps.

Intuitively, the more sample units used for the diagnosis, the longer the detection delay, since new points are sampled periodically. Indeed, the theoretical minimum delay is given by the number of sample units in the passive phase, assuming a history hit is possible, multiplied by the effective sampling interval, i.e., 1.15 seconds in our experiments. Fig. 6a shows both the minimum theoretical delay and the



(a) Sample size $|\mathbf{W_{o.}}| = |\mathbf{W_{i.}}|$ [#] (b) Characterization $C_{outlier}$ [%] (c) CPU limit Th_{CPU} [cores]

Fig. 6. Contention detection delay under different parameter settings of *ACT*.

measured ones partly confirming our intuition. However, one can observe that the decreasing tendency quickly reaches a lower bound. If the samples are too small, the clusters are poorly characterized, hence diagnosis failures are more likely. The increase in extra delay with fewer sample units introduced by failed and restarted diagnoses can be clearly seen from the increasing relative distance to the theoretical minimum, i.e., overhead, across all three Wikimedia components. Taking Apache as an example, the relative overhead grows from 48 percent, corresponding to approximately 1 extra diagnosis every 2, to 230 percent, corresponding to approximately 2 extra diagnoses for each. Similar results hold for the other two components. Combining these results with our previous analysis where a sample size is of 15 or 20 units gives a good trade-off between detection speed and accuracy.

A similar trend is shown in Fig. 6b. Increasing the outlier threshold, increases the probability of failed diagnoses, which in turn increases the detection delay since multiple diagnosis cycles are necessary to raise the first alarm. However, here the influence is lower and the average overhead across different values of $C_{outlier}$ changes from 53 to 86 percent for Apache and from ≈ 60 to ≈ 170 percent for Memcached and MySQL which experience less contention due to lower resource usages.

Finally, considering throttling, the detection delay stays stable as long as the CPU capping limit is low enough for the detection rate not to collapse (see Fig. 6c).

Parameter Tuning. The detection delay is mostly dependent on the number of sample units collected and the success/failure of a diagnosis. The best trade-off is obtained with $|\mathbf{W_{outlier}}| = |\mathbf{W_{isolation}}| = 20$ sample units, $Th_{CPU} = 0.05$ cores and $C_{outlier} = 50\%$. In our tests this translates into a detection delay of 40 seconds and a detection rate never below 90 percent.

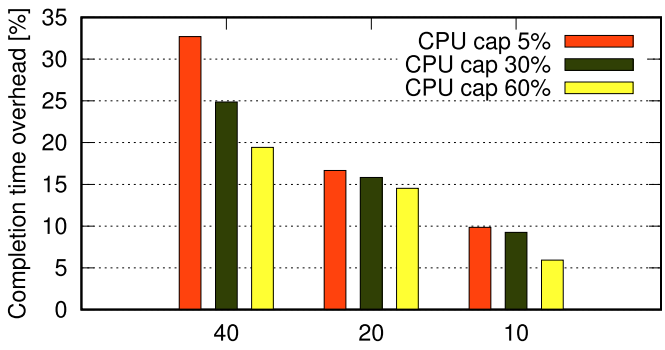


Fig. 7. Average completion time overhead of throttled PARSEC neighboring VM. *X*-axis: Sample size $|\mathbf{W_{o.}}| = |\mathbf{W_{i.}}|$.

6.3.3 Throttling Overhead

We quantify the throttling overhead as the increase in completion time experienced by the neighboring batch jobs. More precisely, we concentrate on the longer running PARSEC workloads. For each experiment run, we quantify the relative time loss for each workload as the difference between the mean execution time during throttled and non-throttled intervals divided by the mean non-throttled execution times. Fig. 7 shows the mean results across the three PARSEC workloads while varying the sample size and the CPU capping limit. We see a clear trend with respect to both. A larger isolation sample directly translates into longer throttling intervals to collect the sample units, hence higher overheads, while shrinking the isolation sample size shrinks the throttling time and overhead. In the case of the CPU capping limit, the opposite is true. Higher CPU limits reduce the performance impairment. Finally, the time loss is stable with respect to $C_{outlier}$, since the parameter has no direct influence on the throttling interval (not shown due to space issues).

Even in the worst case, i.e., highest number of sample units and lowest CPU capping limit, the direct overhead is limited to 33 percent, while for the more reasonable parameter values, i.e., $|\mathbf{W_{outlier}}| = |\mathbf{W_{isolation}}| = 20$ and $Th_{CPU} = 0.3$ cores, the overhead drops to 15 percent. Moreover one must take into consideration that this overhead is only incurred when *ACT* executes the active throttle phase. These phases are a small fraction of the overall diagnosis cycles as shown in Table 3. In most experiments, less than 10 percent of detection cycles rely on the active diagnosis phase. Only in the case of high outlier threshold $C_{outlier}$ on Memcached did *ACT* have difficulties. The reason is that Memcached creates a light contention which is difficult to identify with a high threshold. Combining the direct overhead with the fraction of active diagnoses, the impairment across all intervals is even smaller: 1.1 percent on average and 9.24 percent in the worst case. Moreover, the active diagnosis phases mostly happen at the system startup, when no history table is yet available (see also Section 6.5). Thus, the longer the experiment runs, the smaller the fraction of active diagnosis phases, and the lower the average impact.

TABLE 3
Active Diagnosis Ratio

	$\lvert\mathbf{W_{o.}}\rvert = \lvert\mathbf{W_{i.}}\rvert$ [#]			Th_{CPU} [cores]			$C_{outlier}$ [%]		
	10	20	40	0.05	0.3	0.6	50%	60%	70%
Apache	0.05	0.03	0.07	0.07	0.06	0.10	0.07	0.06	0.05
Memc.	0.03	0.04	0.06	0.06	0.03	0.10	0.06	0.09	0.28
MySQL	0.02	0.04	0.08	0.08	0.11	0.07	0.08	0.03	0.07

TABLE 4
Contention Statistics for Six Workloads

	Det. rate	False alarm	Det. delay [sec]
Data serving	0.94	0.02	40.4
Data caching	0.98	0.00	46.5
Media streaming	0.88	0.02	45.3
Wiki Apache	1.00	0.01	36.9
Wiki Memcached	1.00	0.00	39.5
Wiki MySQL	1.00	0.00	36.2

Parameter Tuning. The throttling overhead is strictly tied to the active throttle phase. Hence the important parameters are $|\mathbf{W_{isolation}}|$ and Th_{CPU}. Considering the tradeoff with the best values for detection we opt for $|\mathbf{W_{isolation}}| = 20$ and $Th_{CPU} = 0.3$ which translates into a throttle overhead of 15 percent. We note again that this overhead is paid only in the few cases requiring the active diagnosis.

6.4 Effectiveness of *ACT* for Cloudsuite

In addition to Wikimedia, we further evaluate *ACT* on three latency-sensitive benchmarks in Cloudsuite [22]: data-serving, data-caching and media-streaming. Here, we configure *ACT* with the best known parameters taken from Wikimedia experiments: $|\mathbf{W_{outlier}}| = |\mathbf{W_{isolation}}| = 20$ sample units, $Th_{CPU} = 0.3$ cores and $C_{outlier} = 50\%$. The testbed and experimental methodology is identical to the one presented in previous sections. We summarize the results obtained from Cloudsuite in Table 4.

On the one hand, the false alarm rate is kept way below 10 percent for all the workloads, again proving the effectiveness of the false positive filtering step. On the other hand, the detection rate drops to 94 percent for data-serving and 98 percent for data-caching. Moreover, the detection delay for these workloads is slightly higher than the Wikimedia nodes. This is because the parameters of *ACT* are optimally tuned for Wikimedia. Hence, the diagnosis for Cloudsuite tends to fail to form the clusters in the second phase and thus *ACT* needs to collect more samples from different intervals. Moreover, one can also observe the lowest detection rate of 88 percent at the media-streaming benchmark. Media-streaming experiences light contention on the memory hierarchy and no contention at all with the network and disk, while being very sensitive to CPU time sharing. If its access to the CPU is impaired, the workload packet transmission latency is affected. The relative lower detection rate indicates that the current metrics collection of *ACT* is not sufficient to capture the CPU time contention.

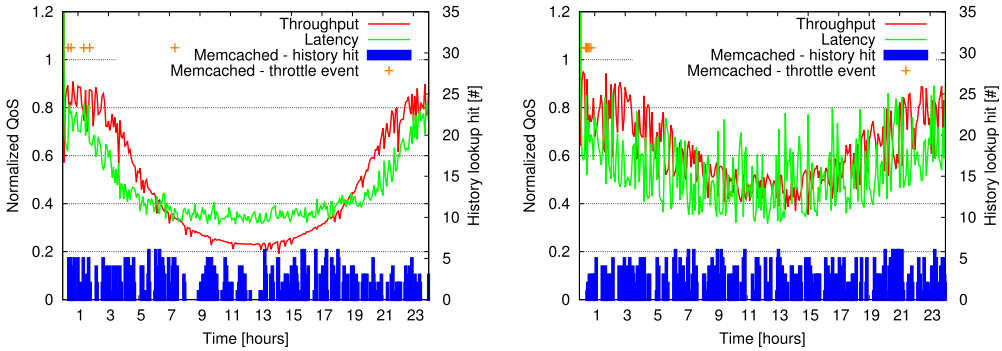
6.5 Long-Running Time-Varying Workload

The object here is to asses how *ACT* performs in realistic scenarios where request rates are time varying. We replace the synthetic constant full load of previous experiments with the load experienced by a real Wikimedia frontend [25] scaled to the capacity of our system. Specifically, the load follows the typical sinusoidal day-night pattern over 24 hours and we scale the intensity such that the offered load oscillates between 90 and 35 percent of the system capacity. In parallel we spawn random batch workloads (both PARSEC and Cachebench) as in our previous experiments, but also consider two VM sizes to create different degrees of contention: i.e., VMs with 2 virtual cores and 2 GB of memory (low contention), and VMs with 8 virtual cores and 8 GB of memory (high contention). In both, we set $|\mathbf{W_{outlier}}| = |\mathbf{W_{isolation}}| = 20$ sample units, $Th_{CPU} = 0.3$ cores and $C_{outlier} = 50\%$ based on the results of the sensitivity analysis.

Results are shown in Fig. 8. We report the measured QoS of Wikimedia, in terms of normalized throughput and latency, together with the outcome of *ACT* on the Memcached server, in terms of alarm rate and throttling events. We only report the outcome of one server for readability and chose Memcached because it has the lowest resource usage making the contention most ambiguous and thus stressing the detection algorithm the most.

From the QoS metrics one can easily infer the level of contention. In the low contention case, shown in Fig. 8a, the client QoS has only small spikes. Moreover, one can see the influence of different Wikimedia loads. Indeed, the spikes are mainly concentrated in the peak load regions and have larger amplitudes. Similar results hold in the high contention case, shown in Fig. 8b, but with overall larger spikes. The clearer the contention, the more contention alarms are raised within a contention interval. Thus, a high density of events, shown by the histograms at the bottom, means that the contention is more obvious and vice-versa. Indeed in Fig. 8b the contention hit density over time is more regular due to the higher degree of contention.

Looking at the throttling events, shown by the crosses, one can see that few contention events are detected using the active diagnosis phase and mainly at system startup. Indeed, after an initial learning phase to fill the history lookup table, most detection cycles avoid the throttle phase.



(a) Low contention (b) High contention

Fig. 8. 24-hour Wikimedia load with random batch jobs. Throughput and latency are normalized to 38 req/s and 330 ms.

TABLE 5
Parallel Detection Analysis of *ACT* on Five Pairs of Collocated Latency-Sensitive VMs

	Det. rate	False alarm	Det. delay [sec]
Wiki Apache + Data serv.	0.98	0.04	50.2
Wiki Memc. + Data serv.	0.98	0.06	61.8
Wiki MySQL + Data serv.	0.83	0.02	56.8
Media stream. + Data serv.	0.96	0.01	60.9
Data caching + Data serv.	0.84	0.11	85.3

Comparing the low and high contention case, the latter allows the table to be filled out faster, i.e., the throttling events are all found within the first hour against the low contention case where the last one is found after 7.3 hours.

6.6 Parallel Analysis

We present how *ACT* can detect resource contentions for multiple latency-sensitive VMs in parallel. To such an end, we consider the scenario where multiple latency-sensitive applications are collocated with PARSEC on the same physical server. For each physical node, we first place two VMs from those four applications such that the contention among them is minimized. Via extensive experiments, we observe that a lowly loaded data-serving benchmark (roughly 20 percent utilization) from Cloudsuite can be collocated with any other applications without causing much contention. Therefore, we place data-serving neighbor VMs to create the five VM pairs shown in Table 5. We then randomly spawn PARSEC on each physical node to inject the emulated contention as in our previous experiments. One *ACT* runs on each physical server and conducts detection analysis for collocated latency-sensitive VMs simultaneously. We summarize how *ACT* can detect resource contention caused by PARSEC across each pair in Table 5, particularly in terms of detection rate, false alarm rate, and detection delay.

The detection rate shown in Table 5 is around 95 percent for data-serving paired with either wiki-Apache, wiki-memcached or media-streaming, suggesting that *ACT* can handle parallel analysis for multiple VMs well. However, the detection rate drops to around 84 percent for the pairs of wiki-MySQL or data-caching, as *ACT* tends to incorrectly label contention as phase-change due to the following two observations. First, wiki-MySQL does not reach the steady-state, violating the assumption made in *ACT*. In contrast to experiments in the previous sections, any of the three Wikimedia VMs can be throttled while *ACT* conducts the active throttling analysis for a collocated VM. As soon as any of the Wikimedia VMs gets throttled, the loads at the other Wikimedia components are also affected and thus the loads fluctuate without staying at steady-state. Second, for data-caching under low usage, it is difficult for *ACT* to differentiate contention and isolation samples. These two observations are further amplified by the history lookup. Once a contention cluster is falsely labelled as phase change during the active diagnosis, *ACT* automatically wrongly labels similar contention cases as phase change during the passive diagnosis.

The average value of the false positive rate shown in Table 5 slightly increases, compared to the scenarios where *ACT* only needs to detect contention for a single latency-sensitive VM presented in previous sections. In the collocated experiments, as both latency-sensitive VMs run at low load to avoid cross contention, the impact of contention caused by PARSEC is thus low. As such, the difference between the contention and normal samples is minimal and diagnosis of *ACT* becomes more ambiguous. This observation is particularly obvious for the pair of data-caching and data-serving.

Compared to the non-collocated case, the time spent to detect the contention, so-called contention delay, is much higher. The main reason is again the low impact of PARSEC inference on lowly utilized latency-sensitive VMs. Thus, *ACT* tends to fail its diagnosis. Another reason is that, with our implementation of the parallel diagnosis, the window used to collect the outlier sample has been purposely increased by 30 percent (25 sample units instead of 20) with respect to the single VM analysis, to increase the detection accuracy.

7 Related Work

Most related work focuses on interference characterization. It can be differentiated by the types of input data, diagnosis overhead, and detection strategy.

Input Data. To detect interference requires both hypervisor and application performance metrics. While hypervisor metrics are readily available to cloud providers, application performance metrics, such as job completion time or request latency, are hidden inside the VM. On the one hand, some prior art [9], [10], [19], [26] explicitly ask such information to be provided. On the other hand, low level metrics such as IPC or CPI can help to predict client side performance [8], [11], [12], [13], [22], [27] or detect phases [6], [7] without the need for explicit feedbacks. Even though IPC can be used for transactional workloads [28], it is not in general applicable [18]. We approach the problem focusing on contention rather than interference, thereby removing the issues of application performance metrics.

Diagnosis Overhead. Completely passive diagnoses are possible, but require more information than the active ones. [11], [27] rely on a very large amount of statistical information gathered on a per application and per platform type. Alternatively, a human operator manually provides anomalous VM signatures in [14]. Active approaches gather the ground truth about anomalies by running the VMs in isolation on dedicated platforms [8], [12], [13], [15]. We take the approach of isolating the VMs by limiting the resource allocation of neighboring VMs directly on production nodes without the overhead of VM migration/cloning or the need for dedicated platforms. A similar approach is taken by [10], but with the objective of finding the optimal VM resource allocation rather than contention detection.

Detection Strategy. Proactive approaches either estimate interference between VMs by being aware of their sensitivity profile [12] or use predictive techniques to trigger an action before interference occurs [9], [26]. Reactive solutions [10], [11], [13], [14], [29] try to characterize the workloads in an on-line fashion and react to behavior anomalies. Our approach is both reactive, since we detect contention when it occurs, and proactive, since detected low contention levels anticipate interference. Another category of related work is simple off-line interference characterization of workloads either running inside VMs or directly on bare metal [27], [30], [31]. We rely on similar metrics to feed our detection system, but strictly in an on-line fashion.

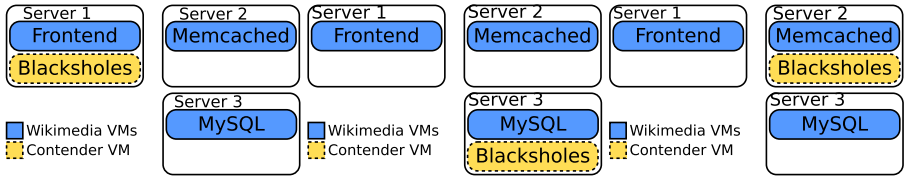


Fig. 9. Example of migrating a batch workload to alleviate contention: Initial deployment (a) and the two possible deployments after migration (b) and (c).

8 Discussion: Resolving Contention

While the focus of *ACT* is to detect the contentions, particularly the transient ones, here we provide a discussion on how to leverage the findings of *ACT* in mitigating the contention. Such a resource contention can be alleviated by various resource management policies, such as changing the VM consolidation plan [5], replicating the same jobs on multiple servers [32] or migrating VMs to different hosts. We specifically present an example of migrating VMs from the contended physical host to the non-contended one. We first deploy Wikimedia together with one Blachscholes benchmark from PARSEC on three servers as in Fig. 9a. When Blackscholes activates the contention on server 1, this results in a 16 percent degradation in the page latency of Wikimedia. Once this contention is detected by *ACT*, we can migrate Blackscholes (to not disrupt the latency-sensitive Wikimedia application) to either server 2 or server 3. Migrating Blackscholes to the server 2 that hosts the MySQL VM reduces the latency degradation to 2.2 percent (see Fig. 9b). In contrast, migrating Blackscholes to server 3 that hosts the Memcached VM even slightly improves the latency by 2.6 percent (see Fig. 9c). These results show that migration is an effective means to counter performance degradation from resource contention and reflects well the observations from Section 6.2. However, the performance difference between migrating to server 2 or server 3 also underlines the difficulty in resolving contention. To optimally mitigate the performance degradation, it is critical that one can translate the low level resource contention into high-level application performance for all possible workload consolidation plans. It is our future work to explore performance modeling techniques to predict, resolve, and optimize collocated application VMs in highly virtualized environments.

9 Conclusion

Virtualization in cloud computing aims to increase the overall system utilization but raises concerns over VM performance isolation. In this paper, we propose an on-line algorithm, *ACT*, to monitor and detect the problem of shared resources contention for VMs of interest by only leveraging low-level metrics and thus guaranteeing the tenant's business confidentiality. At the core is *ACT*, a three-phase contention detection algorithm, that consists of an alarm, passive clustering and active throttle phase involving different tradeoffs of computation complexity and diagnosis accuracy. Via throttling neighboring VMs, *ACT* can accurately discern the contention caused by neighbors from internal program changes. We extensively evaluate *ACT* for latency-sensitive applications, namely Wikimedia and Cloudsuite, hosted on a cloud testbed where different batch-like applications are collocated. We exhaustively tune the parameters of *ACT* against different system scenarios, including time-varying workloads and collocated VMs. Our evaluation results show that *ACT* detects contention in 90 percent of cases in short contention intervals with a direct performance impairment to the collocated VMs as low as 15 percent which can be easily amortized over time.

Acknowledgments

This work has been partly funded by SNSF projects 407540_167266 and 200021_141002.

References

[1] R. Birke, A. Podzimek, L. Y. Chen, and E. Smirni, "State-of-the-practice in data center virtualization: Toward a better understanding of VM usage," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2013, pp. 1–12.

[2] L. Y. Chen, D. Ansaloni, E. Smirni, A. Yokokawa, and W. Binder, "Achieving application-centric performance targets via consolidation on multicores: Myth or reality?" in *Proc. 21st Int. Symp. High-Performance Parallel Distrib. Comput.*, 2012, pp. 37–48.

[3] M. Björkqvist, S. Spicuglia, L. Y. Chen, and W. Binder, "QoS-aware service VM provisioning in clouds: Experiences, models, and cost analysis," in *Proc. Int. Conf. Service-Oriented Comput.*, 2013, pp. 69–83.

[4] M. Björkqvist, L. Y. Chen, and W. Binder, "Opportunistic service provisioning in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 237–244.

[5] C. Wang, B. Urgaonkar, A. Gupta, L. Y. Chen, R. Birke, and G. Kesidis, "Effective capacity modulation as an explicit control knob for public cloud profitability," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2016, pp. 95–104.

[6] A. S. Dhodapkar and J. E. Smith, "Comparing program phase detection techniques," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2003, pp. 217–227.

[7] P. Nagpurkar, C. Krintz, M. Hind, P. F. Sweeney, and V. T. Rajan, "Online phase detection algorithms," in *Proc. Int. Symp. Code Generation Optimization*, 2006, pp. 111–123.

[8] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, "DeepDive: Transparently identifying and managing performance interference in virtualized environments," in *Proc. USENIX Annu. Tech. Conf.*, 2013, pp. 219–230.

[9] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "PREPARE: Predictive performance anomaly prevention for virtualized cloud systems," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 285–294.

[10] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 237–250.

[11] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI2: CPU performance isolation for shared compute clusters," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 379–391.

[12] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proc. Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2013, pp. 77–88.

[13] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "DejaVu: Accelerating resource allocation in virtualized environments," *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 423–436, 2012.

[14] B. Sharma, P. Jayachandran, A. Verma, and C. Das, "CloudPD: Problem determination and diagnosis in shared dynamic clouds," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2013, pp. 1–12.

[15] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, pp. 22:1–22:14.
[16] M. Björkqvist, N. Gautam, R. Birke, W. Binder, and L. Y. Chen, "Optimizing for tail sojourn times of cloud clusters," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, pp. 1–14, 2015, doi: 10.1109/TCC.2015.2474367.
[17] A. Eldin, et al., "How will your workload look like in 6 years? analyzing wikimedia's workload," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2014, pp. 349–354.
[18] A. R. Alameldeen and D. A. Wood, "IPC considered harmful for multiprocessor workloads," *IEEE Micro*, vol. 26, no. 4, pp. 8–17, Jul./Aug. 2006.
[19] J. Mukherjee, D. Krishnamurthy, J. Rolia, and C. Hyser, "Resource contention detection and management for consolidated workloads," in *Proc. IFIP/IEEE Int. Symp. Integrated Netw. Manage.*, 2013, pp. 294–302.
[20] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, Jan. 2011.
[21] D. C. Montgomery, *Introduction to Statistical Quality Control*. Hoboken, NJ, USA: Wiley, 2007.
[22] M. Ferdman, et al., "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 37–48, 2012.
[23] P. J. Mucci, K. London, and P. J. Mucci, "The cachebench report," Univ. Tennessee, Knoxville, TN, USA, 1998.
[24] N. Gunther, S. Subramanyam, and S. Parvu, "Hidden scalability gotchas in memcached and friends," in *Proc. Velocity Web Performance Operations Conf.*, 2010. [Online]. Available: https://conferences.oreilly.com/velocity/velocity2010/public/schedule/detail/13046
[25] W. project, "Wikimedia grid report," 2015. [Online]. Available: http://ganglia.wikimedia.org/latest/
[26] D. J. Dean, H. Nguyen, and X. Gu, "UBL: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2012, pp. 191–200.
[27] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, "Measuring interference between live datacenter applications," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, 2012, pp. 51:1–51:12.
[28] T. Wenisch, R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. Hoe, "SimFlex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, Jul./Aug. 2006.
[29] J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "Contention aware execution: Online contention detection and response," in *Proc. 8th Annu. IEEE/ACM Int. Symp. Code Generation Optimization*, 2010, pp. 257–265.
[30] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Proc. IEEE Int. Symp. Performance Anal. Syst. Softw.*, 2007, pp. 200–209.
[31] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2011, pp. 248–259.
[32] R. Birke, J. Pérez, Q. Zhan, M. Bjoerkqvist, and L. Chen, "Power of redundancy: Designing partial replication for multitier applications," in *Proc. IEEE INFOCOM*, 2017, pp. 271–279.

Joel Vallone received the master's degree from EPFL. He is a software engineer at a Swiss IT Consulting Firm. He is a student member of the IEEE.

Robert Birke received the PhD degree in electronics and communications engineering from the Politecnico di Torino, Italy. He is at IBM Research Zurich Lab. His research interests include the broad area of virtual resource management for large-scale datacenters, including network design, workload characterization, and big-data application optimization. He has published more than 50 papers at venues related to communication and system performance, e.g., SIGCOMM, SIGMETRICS, FAST, INFOCOM, and the *IEEE Journal on Selected Areas in Communications*. He is a senior member of the IEEE.

Lydia Y. Chen received the PhD degree in operations research and industrial engineering from Pennsylvania State University. She is a research staff member at the IBM Zurich Research Lab, Zurich, Switzerland. Her research interests include big data and cloud systems. She has published more than 50 papers in international conferences and journals. She has lead and participated in Swiss National Science Foundation and European FP7 projects. She is a senior member of the IEEE.

Minimizing Data Access Latencies for Virtual Machine Assignment in Cloud Systems

Marzieh Malekimajd and Ali Movaghar, *Senior Member, IEEE*

Abstract—Cloud systems empower the big data management by providing virtual machines (VMs) to process data nodes (DNs) in a faster, cheaper and more effective way. The efficiency of a VM allocation is an important concern that is influenced by the communication latencies. In the literature, it has been proved that the VM assignment minimizing communication latency in the presence of the triangle inequality is 2-approximation. However, a 2-approximation solution is not efficient enough as data center networks are not limited to the triangle inequality. In this paper, we define the quadrilateral inequality property for latencies such that the time complexity of the VM assignment problem minimizing communication latency in the presence of the quadrilateral inequality is in P (polynomial) class. Indeed, we propose an algorithm for the problem of assigning VMs to DNs to minimize the maximum latency among allocated VMs in addition to DNs with their assigned VMs. This algorithm is latency optimal and 2-approximation for networks with the quadrilateral inequality and the triangle inequality, respectively. Besides, the extension of the proposed method can be applied to the cloud elasticity. The simulation results illustrate the good performance and scalability of our method in various known data center networks.

Index Terms—VM assignment, data access, communication latency, elasticity

1 Introduction

There is an increasing need to manage and process large and fast growing amount of data. Cloud computing and its application in big data processing are becoming more popular for big data management while cloud computing offers important benefits like security, scalability and economic benefits [1], [2].

Cloud environments consist of data centers that are connected through networks and provide services. The most important concern in such environments is the resource management. In this regard, the virtualization technology empowers the cloud computing to manage the resources in a multi-user environment while a user receives its requested services on demand from everywhere in any time [3].

Hadoop (an implementation of MapReduce model) is one of the platforms to process huge amount of data that can be implemented in clouds [4] while cloud computing revolutionizes big data management. In Hadoop, VMs are provisioned by cloud systems to process data which are stored in DNs. A good assignment of VMs for a user request leads to provider's benefit and user's satisfaction. However, resource management in cloud systems is not clear due to the scale and uncertainty of future states. Proper models of cloud environment, user requests and resource allocation have to be formulated; then proper algorithms have to be developed for cloud managers.

• *The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran 14588-89694, Iran. E-mail: malekimajd@ce.sharif.edu, movaghar@sharif.edu.*

Manuscript received 7 May 2016; revised 6 July 2017; accepted 27 July 2017. Date of publication 4 Aug. 2017; date of current version 13 Oct. 2020. (Corresponding author: Marzieh Malekimajd.) Digital Object Identifier no. 10.1109/TSC.2017.2735972

Various VM placement problems are considered in the literature regarding cloud services and user criteria [5]. One of the important aspects of a VM placement is communication latency. In services like MapReduce, process nodes communicate with each other and access DNs to fulfill a job. The time of each communication is crucial; thus, the maximum latency between communicating nodes has to be bounded in a VM assignment. The authors in [6] provided the definition of the VM placement problem to optimize communication latencies in cloud systems. The latency of a candidate VM assignment in this problem is defined as the maximum communication latency between any two assigned VMs and any DN with its assigned VM.

In the literature, the VM placement problem (minimizing latency) is addressed by assuming the triangle inequality property on the network topology. The authors in [7] showed that there is no solution better than 2-approximation for the VM assignment problem by considering the triangle inequality. Also, they provided a 2-approximation algorithm to find a VM placement with latency at most twice of the minimum latency.

The triangle inequality has been assumed for the cloud data center networks while there is not a discussion of how accurate this assumption is. Since the triangle inequality affects the bound on latency, it is not necessary to assume it for the VM assignment problem with known data center networks. It is possible to find other reasonable assumptions about the properties of cloud data center networks rather than the triangle inequality. In this way, better theoretical bounds can be found which can lead to better practical solutions. In this paper, a new property, called the quadrilateral inequality, is defined. This property is motivated by the tree-like structure of cloud networks and the Internet [8]. It is worthwhile to mention that the considered VM assignment

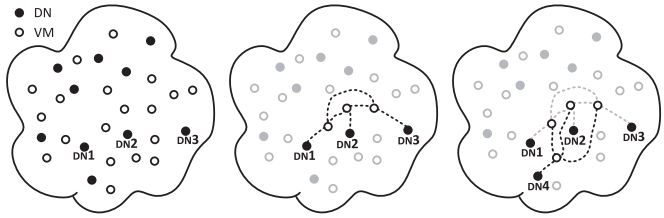
problem with this new property has an optimal solution (minimizing latency) in tree networks.

In cloud systems, it is possible to have unforeseen changes in a requirement or the environment (for example, the addition of a DN or the failure of a VM). Such changes have to be addressed by providing the suitable capacity. This important property of adaptive capacity is called elasticity [9] in which computing resources can be scaled up and down easily by the cloud service provider [10]. Overall, a change in the number of requested VMs has to be handled in a proper VM allocation policy. In this paper, we consider cloud elasticity by handling assignment of VMs to new unmatched DNs. We add new matchings to the existing assignment to minimize the maximum latency of overall VM assignment.

The contribution of this paper is fourfold:

- First, we define a new property for network latencies which is denoted by the quadrilateral inequality. This property holds in tree networks while the topology of several known cloud networks is tree-like. This property resembles the triangle inequality which can be assumed for cloud networks. Under the assumption of the triangle inequality, there is not any polynomial time algorithm for reaching the theoretical bounds of minimum latency for the VM placement problem. However, by considering the quadrilateral inequality, the considered VM assignment problem can have a latency optimal solution.
- Second, we provide a perfect matching algorithm for unbalanced weighted bipartite graphs that finds a matching with minimum diam (maximum edge of the matching). This algorithm finds an assignment of VMs to DNs with a time complexity better than that of the previously known matching algorithms.
- Third, we present the VMtoDNAssignment algorithm to determine an assignment of VMs to DNs with minimum latency. Our proposed algorithm is latency optimal when the quadrilateral inequality is assumed. Also, it is 2-approximation when the triangle inequality is assumed. In addition to the theoretical proof, the simulation results indicate that the proposed method returns a fairly well VM assignment (minimizing latency) in practice. Indeed, these results show that the proposed method finds better assignment (minimizing latency) in comparison to the existing methods.
- Fourth, we extend the VMtoDNAssignment algorithm to be applied to increase the number of requested VMs after the initial VM allocation (elasticity). In the prior studies of the VM assignment problem, a change in number of VMs or DNs (after the initial assignment) is not considered.

The rest of this paper is organized as follows: In Section 2, we describe our problem in addition to the quadrilateral inequality. An algorithm to find a perfect matching with minimum diam in unbalanced weighted bipartite graph is provided in Section 3. Then, we present an algorithm for the VM assignment problem to minimize communication latency in Section 4. In Section 5, our proposed algorithm is extended to be applied to increase the number of requested VMs. The experimental results are discussed in Section 6. In Section 7, we give a brief review of the related work. Finally, Section 8 concludes the paper.



(a) Cloud environment. (b) a VM assignment. (c) a revised assignment.

Fig. 1. Cloud environment with a sample VM assignment and its revision.

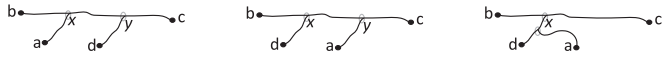
2 Problem of VM Assignment Regarding the Quadrilateral Inequality

In the considered cloud environment, there is a large number of computational nodes and storage nodes that are composed of VMs and DNs, respectively. In this environment, there are jobs that need VMs to access and process a set of DNs; for example, the MapReduce jobs can be pointed out. It is assumed that the set of DNs related to a job is known beforehand while a set of VMs has to be selected and assigned to DNs. In addition, it is possible that a set of DNs or VMs encounters some changes during the job execution. For example, these changes can be caused when a) a new DN is added, b) a DN needs more assigned VMs and c) an assigned VM to a DN goes down. All of these cases can be considered as the addition of a new DN in which the VM assignment of the considered job requires a revision.

Without loss of generality, we assume that the number of needed VMs is equal to the number of DNs for a job. Also, each VM is assigned to exactly one DN while each DN needs exactly one assigned VM. If this assumption does not hold for a VM/DN (that is, a VM can be assigned to more than one DN or a DN needs more than one assigned VM), it is enough to duplicate it in the problem formulation. For example, when a DN needs three assigned VMs, we assume three copies of this DN that each copy needs one VM. Also, it is the same for VMs that can process more than one DN.

In the problem formulation, we consider one request at a time, i.e., one job with a set of DNs. Therefore, several requests at the same time are handled consecutively (not simultaneously) in our model. Also, in the context of the VM assignment problem, most of the studies assumed one request at a time. However, a less optimal solution can be produced by considering (at the same time) requests separately than considering them all together, therefore, we will address considering requests simultaneously in our future work.

In this paper, the problem of VM assignment (for a job) is defined as selecting n VMs (from the set of available VMs) and assigning them to n DNs such that the latency of the assignment of VMs to DNs be minimum. For example, let consider a job J with three DNs DN1, DN2 and DN3 in Fig. 1a. A sample VM assignment for job J is shown in Fig. 1b. The edges of this assignment are shown by dashed lines while the edge with maximum latency determines the latency of the assignment. Further, the problem includes the case of scaling up and down the number of allocated VMs for an existing VM assignment after the initial assignment



(a) Case 1. (b) Case 2. (c) Case 3.

Fig. 2. The paths between four nodes in a tree.

of VMs to DNs. In Fig. 1c, DNS of job J are increased by DN4 while new added edges to the existing VM assignment are shown by black dashed lines. Here, decreasing the number of VMs is clear, however, increasing the number of VMs needs more attention. In the case of increasing the number of VMs, free VMs are selected and assigned to unmatched DNs such that the maximum latency regarding the initial VM assignment (that is, the latency of overall VM assignment) be minimum.

Assumed properties on the communication latencies in the VM assignment problem have an important influence on the proposed algorithms. Therefore, we define a property on the communication latencies which is denoted by the quadrilateral inequality. Assuming this property in the VM assignment problem is an important aspect of the presented algorithm in Section 4. In what follows, first, we provide the definition of the quadrilateral inequality. Second, we provide a theorem on the quadrilateral inequality property in tree networks. Third, we discuss the assumption of the quadrilateral inequality for known data center networks. At last, our considered problem regarding the quadrilateral inequality in comparison to the triangle inequality is discussed.

Definition 2.1. *Let consider four nodes a, b, c and d in a graph. Also, let $d_{u,v}$ denote the distance (or latency) between two nodes u and v. The quadrilateral inequality states that if $\max\{d_{a,c}, d_{d,c}, d_{a,b}, d_{d,b}\} \leq d_{b,c}$, then $d_{a,d} \leq d_{b,c}$. In other words, when the quadrilateral inequality holds, if $d_{b,c}$ is not less than $d_{a,c}$, $d_{d,c}$, $d_{a,b}$ and $d_{d,b}$ then $d_{b,c}$ is not less than $d_{a,d}$.*

Theorem 2.2. *Every tree graph has the quadrilateral inequality property.*

Proof. We show that the quadrilateral inequality is satisfied for any four nodes a, b, c and d in a tree that $\max\{d_{a,c}, d_{d,c}, d_{a,b}, d_{d,b}\} \leq d_{b,c}$. Let consider the paths that connect a, b and c. Because there is not a cycle in a tree, these paths are crossed in exactly one node. If we consider the node d in addition to other nodes, one of the cases in Fig. 2 is resulted. We discussed the case in Fig. 2a while other cases are similar.

We can conclude the following inequalities from the relations between latencies

$$d_{a,c} \leq d_{b,c} \Rightarrow d_{a,x} + d_{x,c} \leq d_{b,x} + d_{x,c} \Rightarrow d_{a,x} \leq d_{b,x} \quad (1)$$

$$d_{d,b} \leq d_{b,c} \Rightarrow d_{d,y} + d_{y,b} \leq d_{b,y} + d_{y,c} \Rightarrow d_{d,y} \leq d_{y,c}. \quad (2)$$

By substituting the inequalities (1) and (2) in $d_{a,d}$, the following inequality is resulted

$$\begin{aligned} d_{a,d} = d_{a,x} + d_{x,y} + d_{y,d} \leq d_{b,x} + d_{x,y} + d_{y,c} \leq d_{b,c} \\ \Rightarrow d_{a,d} \leq d_{b,c}. \end{aligned} \quad (3)$$

Therefore, the quadrilateral inequality is satisfied for any four optional nodes a, b, c and d that $\max\{d_{a,c}, d_{d,c}, d_{a,b}, d_{d,b}\} \leq d_{b,c}$. □

TABLE 1
Notation Definitions

Notation	Definition
$diam$	The edge with maximum weight in a matching.
$d_{u,v}$	The distance between two nodes u and v. Note that $d_{u,v} = d_{v,u}$.
d_e	The latency of the edge e.
d_S	The diameter of the set S.
A VM assignment	a subset of VMs in addition to a matching from DNs to these VMs.
The latency of the assignment of VMs to DNs	The maximum latency among allocated VMs in addition to DNs and their assigned VMs.
The quadrilateral inequality	If $\max\{d_{a,c}, d_{d,c}, d_{a,b}, d_{d,b}\} \leq d_{b,c}$, then $d_{a,d} \leq d_{b,c}$.

Here, we briefly discuss the assumption of the quadrilateral inequality for the cloud networks. Hierarchical, Tree, VL2, Fat-tree and BCube are five known data center networks in cloud systems [11]. If we assume that these networks are symmetric (that is, for a connection, different choices of links and switches at the same level have equal latencies), then it is possible to connect racks in these networks by a tree such that the latency between two optional racks in a tree remains the same as the initial network. In such cases, the mentioned networks have the quadrilateral inequality. However, these networks are not always symmetric and the degree that a network can be approximated to have the quadrilateral inequality is dependent to its structure and its components latencies. The practical performance of assuming the quadrilateral inequality for the known data center networks (regarding the VM assignment problem) is reported in Section 6.

Moreover, it is worthwhile to mention that the quadrilateral inequality is defined to avoid the triangle inequality assumption. Prior studies consider the VM allocation problem with the triangle inequality assumption on latencies [6], [7]. The authors in [7] showed that there is no solution better than 2-approximation in the presence of the triangle inequality. However, the time complexity of this problem with the assumption of the quadrilateral inequality is in P class (it can be solved in polynomial time) since we present a latency optimal algorithm for it in Section 4. Besides, our proposed algorithm is 2-approximation for the assumption of the triangle inequality. Theorems 4.1 and 4.2 show that our proposed algorithm is optimal and 2-approximation for the quadrilateral inequality and the triangle inequality, respectively.

Table 1 summarize the notations and definitions that are used in this paper.

3 Perfect Matching in Unbalanced Weighted Bipartite Graphs

In this section, we present an algorithm to find a perfect matching with minimum diam in an unbalanced weighted bipartite graph. This algorithm is used in next section to provide an assignment of VMs to DNs. Using this algorithm

rather than the existing methods improves the time complexity of VM allocation.

A matching in a graph is defined as a subset of edges such that no vertex has more than one neighbor. Let consider an unbalanced weighted bipartite graph $G := (D; V; E)$ such that the set $D \cup V$ is its vertices ($|D| < |V|$) and the set E is its edges from D to V. A perfect matching in the graph G is a subset of the set E such that each vertex in the set D has exactly one neighbor in the set V. In the context of matching problem, an alternating path is a path in which the edges belong alternatively to the matching and not to the matching. Also, an augmenting path is an alternating path that starts from and ends on unmatched vertices. The cardinality of a considered matching M is increased by taking the symmetric difference of an augmenting path with M.

Our matching algorithm is presented in Section 3.1. The main idea of this algorithm is to construct a particular tree to find augmenting paths faster. Such trees are constructed iteratively by adding vertices from the set $D \cup V$ and the edges from the set E. An example of the MinMatching Algorithm, its correctness and its time complexity are given in Sections 3.2, 3.3, and 3.4, respectively.

3.1 MinMatching Algorithm

The MinMatching algorithm takes as input the unbalanced weighted bipartite graph $G := (D; V; E)$ which D is smaller part of vertices and V is the bigger one. It is assumed that the set $E = \{d_{i,j} | i \in D \text{ and } j \in V\}$ is sorted ascending (otherwise, this set can be sorted in time $O(|E| \times log|E|)$). The algorithm reports as output a perfect matching M with minimum diam (which it is from the set D to the set V).

The MinMatching algorithm constructs a tree T and uses it to find augmenting paths in the graph G. Initially, the tree T consists of a vertex r_d as its root (this vertex is not in the set $D \cup V$) and all vertices in the set D are added as children of the vertex r_d (note that the vertices in the set V are not in the tree T at first). Subsequently, a set F is considered to construct the tree T (which is empty at first and is a subset of the set E).

Iteratively, an edge of the set E (next smallest edge) is added to the set F. In each iteration, the development of the tree T is examined by checking the new added edge to the set F. If the new edge has exactly one end in the tree T, then its other end is added to the tree T. Whenever a new vertex can be added to the tree T, an algorithm similar to DFS is called to add other vertices that can be reached (regarding the set F) from this new vertex and are not in the tree T. Whenever a new vertex (and probably some other new vertices which are reached from it) is added to the tree T, the existence of an augmenting path is checked. If there is an unmatched vertex from the set V in the Tree T, then there is an augmenting path in T. Therefore, the new added subtree is checked to find an unmatched vertex from the set V.

The algorithm MatchUpdate is called to add the reachable vertices from the new added vertices while it checks the existence of an augmenting path simultaneously. When this algorithm finds an augmenting path, it stops the process of adding the vertices and updates the found augmenting path (switch the edges along the augmenting path from in to out of the the matching M while the cardinality of the matching M is increased). After updating an augmenting path, the tree T is reconstructed and the next iteration is run. Finally, when all vertices in the set D have one matched vertex in the set V, the algorithm concludes. The details of the MinMatching algorithm and the MatchUpdate algorithm are illustrated in Algorithms 1 and 2, respectively.

Algorithm 1. $MinMatching$ Algorithm

Require: An unbalanced weighted bipartite graph $G := (D; V; E)$
1: $T \leftarrow$ a tree with root r_d and all $d \in D$ as children of r_d
2: $M \leftarrow null$, $F \leftarrow null$
3: **while** there is an unmatched vertex in D **do**
4: $(i, j) \leftarrow$ next smallest member of E $(i \in D, j \in V)$
5: $F \leftarrow F \cup \{(i, j)\}$
6: **if** $i \in T$ **and** $j \notin T$ **then**
7: $T \leftarrow T \cup \{j$ as a child of $i\}$
8: **if** $MatchUpdate(j)$ equals to 1 **then**
9: $T \leftarrow$ a tree with root r_d and all unmatched $d \in D$ as children of r_d
10: DFSlist $\leftarrow$ all unmatched $d \in D$
11: **while** DFSlist is not $null$ **do**
12: $df \leftarrow$ next member of DFSlist
13: $Ne \leftarrow \{$neighbors $(df)\} - T$ (according to the set F)
14: **for** each $n \in Ne$ **do**
15: $T \leftarrow T \cup \{n$ as a child of $df\}$
16: **if** n has a match **then**
17: add its match to DFSlist
18: add its match to T as a child of n
19: **end if**
20: **end for**
21: **end while**
22: **end if**
23: **end if**
24: **end while**
25: **return** matching M

Algorithm 2. $MatchUpdate$ Algorithm

Require:a vertex j
1: **if** j is unmatched (i.e., there is an augmenting path from j) **then**
2: update the matching M regarding the found augmenting path
3: **return** 1
4: **end if**
5: $i \leftarrow$ match(j), $T \leftarrow T \cup \{i$ as a child of $j\}$
6: **for** each $k \in \{$neighbors$(i)\} - \{j\}$ (according to the set F) **do**
7: **if** $k \notin T$ **then**
8: $T \leftarrow T \cup \{k$ as a child of $i\}$
9: **end if**
10: **if** $MatchUpdate(k)$ equals to 1 **then**
11: **return** 1
12: **end if**
13: **end for**
14: **return** 0

3.2 An Example of the MinMatching Algorithm

Here, we provide an example of the MinMatching algorithm. The input sets D, V and weights of the edges in the set E are illustrated in Fig. 3a. The different states of the tree T and the matching M (regarding the While statement in Line 3 of the

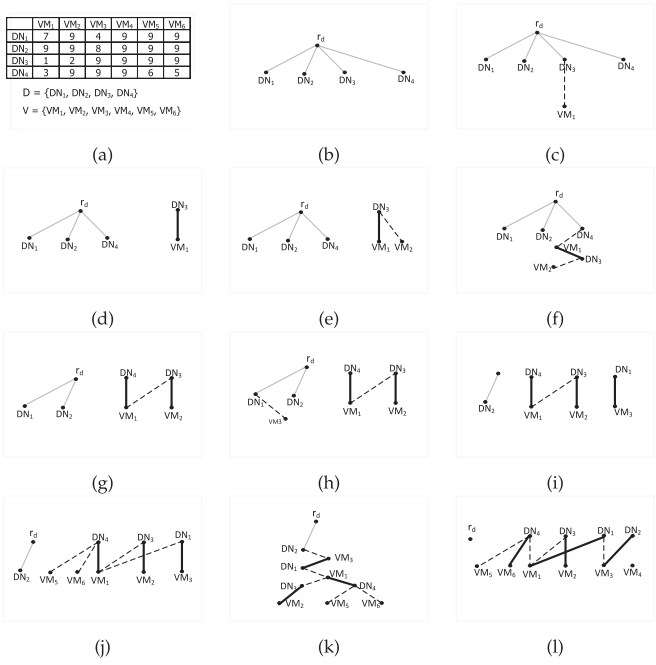


Fig. 3. An example of the MinMatching algorithm.

MinMatching algorithm) are illustrated in Figs. 3b, 3c, 3d, 3e, 3f, 3g, 3h, 3i, 3j, 3k, and 3l. In these figures, the tree T is drawn in left, which is rooted by the node r_d and the edges sof the matching M are determined by solid lines. In following, the steps of the considered example (regarding Figs. 3b, 3c, 3d, 3e, 3f, 3g, 3h, 3i, 3j, 3k, and 3l) are explained:

The initial tree T is constructed in Line 3 of the MinMatching algorithm as shown in Fig. 3b. The While statement in Line 3 of the MinMatching algorithm considers the edges regarding their weights. These edges are $\{(DN_3, VM_1), (DN_3, VM_2), (DN_4, VM_1), (DN_1, VM_3), (DN_4, VM_6), (DN_4, VM_5), (DN_1, VM_1), (DN_2, VM_3), (DN_1, VM_2), \ldots\}$.

- The edge (DN_3, VM_1) is added to the set F (Line 5): This edge is added to the tree T (Line 8) as shown in Fig. 3c. MatchUpdate(VM_1) (which is called in Line 9) adds the edge (DN_3, VM_1) to the matching M. Then Lines 10-23 reconstruct the tree T. The result is shown in Fig. 3d.
- The edge (DN_3, VM_2) is added to the set F (Line 5): The IF statement in Line 7 does not hold. The result is shown in Fig. 3e.
- The edge (DN_4, VM_1) is added to the set F (Line 5): Line 8 and MatchUpdate(VM_1) reconstruct the tree T as shown in Fig. 3f. The last recursive call of the MatchUpdate algorithms is MatchUpdate(VM_4) that leads to find the augmenting path $\{(DN_4, VM_1), (VM_1, DN_3), (DN_3, VM_2)\}$. The update of this augmenting path in addition to the reconstruction of tree T (Lines 10-23) is shown in Fig. 3g.
- The edge (DN_1, VM_3) is added to the set F (Line 5): This edge is added to the tree T (Line 8) as shown in Fig. 3h. Again, the MatchUpdate algorithm adds the edge (DN_1, VM_3) to matching M and Lines 10-23 reconstruct the tree T. The Result is shown in Fig. 3i.
- The edges (DN_4, VM_6), (DN_4, VM_5) and (DN_1, VM_1) are added to set F by the While statement (Line 3): The tree T and matching M do not change. The result is shown in Fig. 3j.
- The edge (DN_2, VM_3) is added to the set F (Line 5): Line 8 and MatchUpdate(VM_3) reconstruct the tree T as shown in Fig. 3k. The last recursive call of the MatchUpdate algorithms is MatchUpdate(VM_6) that leads to find the augmenting path $\{(DN_2, VM_3), (VM_3, DN_1), (DN_1, VM_1), (VM_1, DN_4), (DN_4, VM_6)\}$. The update of this augmenting path in addition to the reconstruction of tree T (Lines 10-23) is shown in Fig. 3l.

Now, all vertices in the set D are matched and the MinMatching algorithm is finished. Overall, the edges of the perfect matching with minimum diam are (DN_1, VM_1), (DN_2, VM_3), (DN_3, VM_2) and (DN_4, VM_6).

3.3 Correctness of the MinMatching Algorithm

Let the perfect matching with minimum diam be the set $M_{Opt} = \{e_1 \leq e_2 \leq \cdots \leq e_{|D|})\}$ and the output of the MinMatching algorithm be the set M_{alg}. Also, let M^c_{alg} be the computed matching in Line 2 of the MatchUpdate algorithm when $F = F_c = \{e | e \in E \text{ and } d_e \leq d_{e_c}\}$ (that is, after the edge $e_c \in M_{opt}$ is added to the set F in Line 5 of the MinMatching algorithm). To show the correctness of the MinMatching algorithm, it is enough to show that the set $M^{|D|}_{alg}$ has the cardinality $|D|$. In following, we use mathematical induction to show that M^c_{alg} has cardinality at least c ($0 \leq c \leq |D|$).

The base case of the induction: the set M^0_{alg} has cardinality 0 when $c = 0$. The proof is clear.

The inductive step of the induction: the set M^c_{alg} has cardinality at least c if the set M^{c-1}_{alg} has the cardinality at least $c - 1$.

The proof of the induction hypothesis: if $|M^{c-1}_{alg}| \geq c$, then the proof is complete, therefore, we consider the case that $|M^{c-1}_{alg}| = c - 1$. We examine the number of augmenting paths in the graph $G := (D; V; F_c)$. By considering the matching M_{Opt}, there are at least c disjoint augmenting paths in the set F_c when no edge from this set is considered as a matching edge. We iteratively add an edge of the set F_c (regarding the matching M^{c-1}_{alg}) to the matching M^c. When the role (i.e., be in matching or not) of an edge from the set F_c is changed, the number of augmenting paths is decreased by one. That is because when an edge is added to the matching, two disjoint augmenting paths become one augmenting path or an augmenting path is updated to an ordinary path. Hence, at least one augmenting path remains in the set F_c when $c - 1$ edges from the matching M^{c-1}_{alg} in the set F_c are added to the matching M^c_{alg}.

Now, it is enough to show that the remained augmenting path (which is denoted by P_{aug}) leads to a matching with cardinality $|c|$. Let the edge $(d_i, v_i) \in F_c$ be the last edge that is added to the path P_{aug}. This edge is not in the matching M^c_{alg} because it is recently added to the set F_c. The vertex d_i is in the tree T and the vertex v_i is not in the tree T, otherwise, before the addition of this edge, there is an augmenting path that consists of a path from v_i to r_d and a path from v_i to the unmatched vertex v (one end of the path P_{aug}). Hence, the $MatchUpdate(v_i)$ is called in Line 8. The MatchUpdate recursively calls itself regarding the tree T until it reaches the end of the path P_{aug}. Therefore, an unmatched vertex is found that leads to find the augmenting path P_{aug}.

Finally, this path is updated and as a result the cardinality of the matching M^c_{alg} is increased.

Overall, the MinMatching algorithm finds a matching with $|D|$ edges (i.e., a perfect matching) after the edge $e_{|D|}$ is added to the set F. This matching has the diam $d_{e_{|D|}}$, therefore, the output of the MinMatching algorithm is a perfect matching with minimum diam.

3.4 Time Complexity

The time complexity of the MinMatching algorithm is mainly related to the While statement in Line 3 that is iterated at most $|E|$ times. However, to have a better estimation, independent of the number of iterations of the While statement in Line 3, we examine the time complexity of the If statement in Line 8 (i.e., the time complexity of all calls of the MatchUpdate algorithm) and the While statement in Line 11. In following, the time complexity of all calls of the MatchUpdate algorithm and the reconstruction of the tree T (i.e., lines 8-22) are stated.

The MatchUpdate algorithm checks edges (and adds them to the tree T) that have one matched vertex from the set V. This algorithm recursively calls itself until there are no more edges to check or it encounters an edge with one unmatched vertex from the set V. When this algorithm meets an edge that has an unmatched vertex from the set V, this means that an augmenting path is found. The number of edges with matched vertices from the set V is $O(|D|)$. Also, updating an augmenting path is $O(|D|)$. As the tree T is not reconstructed between two updates of augmenting paths, we can conclude that the time complexity of multiple calls of the MatchUpdate algorithm between two updates of augmenting paths is $O(|D|)$ (that is, $O(|D|)$ edges are examined and a path with $O(|D|)$ edges is updated). As there are $O(|D|)$ augmenting paths, all the calls of the MatchUpdate algorithm have the time complexity $O(|D| \times |D|)$.

The MatchUpdate algorithm returns 1 at most $O(|D|)$ times. That is because it returns 1 when it finds an augmenting path and there are $|D|$ augmenting paths. The body of the If statement in Line 8 reconstructs the tree T whenever the MatchUpdate algorithm returns 1. Therefore, the body of the If statement (moreover the While statement in Line 11) is executed $O(|D|)$ times. Indeed, each execution of the While statement in Line 11 has the time complexity $O(|D|)$ because the number of vertices of the tree T is $O(|D|)$. Overall, the time complexity of all executions of the body of the If statement in line 8 is $O(|D| \times |D|)$.

It is concluded that the time complexity of the MinMatching algorithm is $O(|E|)$ because each part of it has the time complexity $O(|E|)$ or $O(|D| \times |D|)$. It is good to note that the time complexity of the MinMatching algorithm is $O(|E|\, log(|E|))$ if the set E is not sorted. As far we know, there is not any algorithm in the literature for the perfect matching with minimum diam and our proposed algorithm is the first one. Also, the best time complexity of the minimum perfect matching in unbalanced weighted bipartite graphs is $O(|D| \times (|E| + |D| \times log(|D|)))$ [12]. However, this algorithm is not helpful in this paper, because the sum of the edges is considered instead of the diam.

4 Assignment of VMs to DNs

In this section, we provide an algorithm to find a set of VMs and assign them to the considered set of DNs such that the latency of the assignment of VMs to DNs (as defined in Table 1) be minimum. Our proposed algorithm which is denoted by the VMtoDNAssignment algorithm, is described in Section 4.1. This algorithm finds an optimal assignment when the quadrilateral inequality holds in the communication latencies. Also, it is a 2-approximation algorithm when the communication latencies satisfy the triangle inequality. The optimality and time complexity of the VMtoDNAssignment algorithm are discussed in Sections 4.2 and 4.3, respectively.

The main idea of the VMtoDNAssignment algorithm is to examine all pairs of VMs as the diameter of the solution. The examination is performed by calculating the best perfect matching from DNs to a subset of VMs with the considered diameter. Overall, the best VM assignment for a job is selected by comparing calculated VM assignments. A VM assignment includes a subset of VMs in addition to a matching from DNs to these VMs.

4.1 VMtoDNAssignment Algorithm

Let the set of available VMs and the set of DNs be denoted by V and D, respectively. The VMtoDNAssignment algorithm examines each pair $(v_i, v_j) \in V \times V$ that d_{v_i,v_j} is less than the latency of the best found VM assignment (which holds in the algorithm and updated after each examination). The examination of the pair (v_i, v_j) is started by calculating a maximal set $s_{max} \subset V$ such that $\{v_i, v_j\} \subset s_{max}$ and $d_{s_{max}} = d_{v_i,v_j}$.

The maximal set s_{max} is calculated by calling the MaximalSet algorithm which takes two nodes (v_i, v_j) and returns the set $\{u | u \in V, d_{u,v_i} \leq d_{v_i,v_j} \text{ and } d_{u,v_j} \leq d_{v_i,v_j}\}$. Note that the output of the MaximalSet algorithm has the maximal cardinality as its diameter is (v_i, v_j). If the set s_{max} has enough number of VMs (i.e., $|s_{max}| \geq |D|$), then the VMtoDNAssignment algorithm calculates an optimal assignment of the set s_{max} to the set D by the MinMatching algorithm. Overall, the examination of the pair (v_i, v_j) leads to a VM assignment (if it exists) that its latency is $maximum(d_{v_i,v_j}, \text{the diam of the matching from } D \text{ to } s_{max})$. The best found VM assignment is updated whenever a new possible VM assignment with better latency is found. Note that a VM assignment consists of a subset of the set s_{max} with cardinality $|D|$ that its nodes are matched to the nodes of the set D. At last, when all pairs $(v_i, v_j) \in V \times V$ are examined, the VMtoDNAssignment algorithm reports the best found VM assignment as output. The details of the VMtoDNAssignment algorithm and the MaximalSet algorithm are illustrated in Algorithms 3 and 4, respectively.

4.2 Optimality

Initially, we prove that the set maxSet in Line 7 of the MaximalSet algorithm is maximal while the edge (v_i, v_j) is its diameter. Subsequently, Theorem 4.1 is asserted that the VMtoDNAssignment algorithm is latency optimal in the presence of the quadrilateral inequality. Moreover, Theorem 4.2 is asserted that the VMtoDNAssignment algorithm is 2-approximation in the presence of the triangle inequality.

The proof of the correctness of the MaximalSet algorithm has two parts. First, there is not a node that it is not in the set $maxSet$ but can be added to it, i.e., the set $maxSet$ is maximal. That is because each node $t_s \in V - maxSet$

increases the diameter of the set $maxSet$ regarding Line 3 of the MaximalSet algorithm. Second, the diameter of the set $maxSet$ equals to d_{v_i,v_j}. Let assume two optional nodes $\{o_1, o_2\} \subset maxSet$. If we consider these two nodes in addition to v_i and v_j regarding the quadrilateral inequality, we can conclude $d_{o_1,o_2} \leq d_{v_i,v_j}$ and, therefore, the diameter of the set $maxSet$ is d_{v_i,v_j}.

Algorithm 3. *VMtoDNAssignment* Algorithm

Require: a set D (DNs), a set V (available VMs) and $\{d_{u,v}|(u,v) \in (D \cup V) \times (D \cup V)\}$
1: $L \leftarrow$ sorted list $\{d_{u,v}|u \in D \text{ and } v \in V\}$
2: candidateSet $\leftarrow$ a random VM assignment
3: **for** each $(v_i, v_j) \in V \times V$ **do**
4: **if** $d_{v_i,v_j} >$ latency of candidateSet **then**
5: continue
6: **end if**
7: $s_{max} \leftarrow$ MaximalSet(v_i, v_j)
8: **if** $|s_{max}| \geq |D|$ **then**
9: $L_{D,s_{max}} \leftarrow$ sorted list $\{d_{u,v}|u \in D \text{ and } v \in s_{max}\}$
10: $M \leftarrow$ MinMatching(D, s_{max}, $L_{D,s_{max}}$)
11: $A \leftarrow$ VM assignment related to M
12: **if** A is better than candidateSet **then**
13: candidateSet $\leftarrow A$
14: **end if**
15: **end if**
16: **end for**
17: **return** candidateSet

Algorithm 4. *MaximalSet* Algorithm

Require:a set V, a pair $(v_i, v_j) \in V$ and $\{d_{u,v}|(u,v) \in V \times V\}$
1: maxSet $\leftarrow null$
2: **for** each $u \in V$ **do**
3: **if** $d_{u,v_i} \leq d_{v_i,v_j}$ and $d_{u,v_j} \leq d_{v_i,v_j}$ **then**
4: maxSet $\leftarrow$ maxSet $\cup \{u\}$
5: **end if**
6: **end for**
7: **return** maxSet

Theorem 4.1. *The VMtoDNAssignment algorithm, in the presence of the quadrilateral inequality, finds an assignment of VMs to DNs such that the latency of this VM assignment be minimum.*

Proof. We assume, by contradiction, that there is an assignment of VMs to DNs with a latency less than the latency of the output of the VMtoDNAssignment algorithm. Note that an assignment of VMs to DNs means that a subset of the set V is selected and their nodes are matched to DNs. Let the best VM assignment and the selected VMs in this VM assignment be denoted by cnt and V_{cnt}, respectively. Two cases for the edge with maximum latency in the VM assignment cnt have to be considered. First, both ends of this edge are VMs. Second, one end is a VM and another one is a DN. Each case is discussed separately in the following.

First case, let consider $(t_1, t_2) \in V \times V$ is the diameter of the VM assignment cnt and, therefore, the latency of cnt equals to d_{t_1,t_2}. When the nodes t_1 and t_2 are examined in Line 3 of the VMtoDNAssignment algorithm, a maximal set s_{max} with the diameter (t_1, t_2) is calculated. The set s_{max} is a superset of the set V_{cnt} due to the maximality of the set s_{max}. In addition, we consider that $(dn, v_{dn}) \in D \times V$ is the diam of the matching from the set D to the set V_{cnt} in the VM assignment cnt. The maximum latency between DNs and their assigned VMs in V_{cnt} (i.e., the diam of the matching) equals to $d_{dn,v_{dn}}$ (which is the latency between the node $dn \in D$ and its matched node $v_{dn} \in V$). The VMtoDNAssignment algorithm finds a perfect matching with diam at most $d_{dn,v_{dn}}$ because there is a perfect matching from the set D to the set V_{cnt} and the MinMatching algorithm in Line 10 reports the optimal matching. Overall, the VMtoDNAssignment algorithm examines at least one VM assignment as good as the VM assignment cnt and, therefore, the contradiction assumption is not true.

Second case, let consider two nodes $s_1 \in V_{cnt}$ and $s_2 \in D$ such that the latency of the VM assignment cnt equals to d_{s_1,s_2} and the edge $(t_1, t_2) \in V_{cnt} \times V_{cnt}$ is the diameter of the set V_{cnt}. When two nodes t_1 and t_2 are examined in Line 3 of the VMtoDNAssignment algorithm, a maximal set including t_1 and t_2 is calculated that is denoted by Sp and it is a superset of the set V_{cnt}. Similar to the prior case, the VMtoDNAssignment algorithm finds the perfect matching with minimum diam from the set Sp to the set D. This matching is not worse than the matching in the VM assignment cnt because any perfect matching for the set V_{cnt} is a perfect matching for the set Sp when Sp is a superset of the set V_{cnt}. Hence, the related matching of the set Sp is not worse than the matching in the VM assignment cnt. Overall, the VMtoDNAssignment algorithm examines at least one set which its VM assignment is as good as the VM assignment cnt and, therefore, the contradiction assumption is not true. □

Theorem 4.2. *The VMtoDNAssignment algorithm, in the presence of the triangle inequality, is a 2-approximation algorithm, i.e., it finds an assignment of VMs to DNs with latency at most twice of the latency of the optimal assignment.*

Proof. Let consider the optimal VM assignment consists of the set $OptSet \subset V$ and the edge (t_1, t_2) is the diameter of the set $OptSet$. All members of the set $OptSet$ are added to the set s_{max} in Line 7 of the VMtoDNAssignment algorithm (i.e., $OpeSet \subset s_{max}$) due to the maximality of the set s_{max}. Hence, the optimal perfect matching from the set D to the set $OptSet$ (which is denoted by MM_{opt}) is a perfect matching from the set D to the output of $MaximalSet(t_1, t_2)$. Therefore, the optimal perfect matching related to the set s_{max} (which is denoted by MM) is not worse than the matching MM_{opt}. The matching MM is examined as a candidate set in Line 12 and, therefore, the output of the VMtoDNAssignment algorithm is equal or better than it. Finally, it is enough to show that the latency of the VM assignment related to the matching MM is at most twice the latency of the optimal VM assignment. To show this, we verify the latency between two optional nodes u and v in the VM assignment related to the matching MM.

- First case: When both of nodes u and v are VMs ($\{u\ v\} \subset s_{max}$), we have $d_{u,t_1} \leq d_{t_1,t_2}$ and $d_{v,t_1} \leq d_{t_1,t_2}$ regarding Line 3 of the MaximalSet

algorithm. Moreover, we have $d_{u,v} \leq d_{u,t_1} + d_{v,t_1}$ regarding the triangle inequality between three nodes u, v and $t1$. By putting together these three inequalities, we have

$$d_{u,v} \leq d_{u,t_1} + d_{v,t_1} \leq d_{t_1,t_2} + d_{t_1,t_2} \leq 2 \times d_{t_1,t_2}. \quad (4)$$

- Second case: $d_{u,v}$ ($(u,v) \in MM$) is equal or less than the diam of the matching MM_{opt} because we showed that the diam of MM_{opt} is not less than the diam of the matching MM. Overall, the maximum latency related to the output of $MaximalSet(t_1,t_2)$ and the matching MM is not greater than $2 \times d_{t_1,t_2}$. Therefore, the VMtoDNAssignment algorithm returns a 2-approximation VM assignment of the optimal VM assignment. □

4.3 Time Complexity

The list L in Line 1 of the VMtoDNAssignment algorithm is sorted in time $O(|E|log(|E|))$. The For loop in Line 3 of the VMtoDNAssignment algorithm is iterated at most $O(|V|^2)$. The MaximalSet algorithm and the MinMatching algorithm are called in each iteration of this loop. The time complexity of the MaximalSet algorithm is $O(|V|)$ because the For loop in Line 2 of the MaximalSet algorithm is iterated $O(|V|)$ times. The list $L_{D,s_{max}}$ in Line 9 of the VMtoDNAssignment algorithm can be sorted in time $O(|E|)$ with regard to the sorted list L. The time complexity of the MinMatching algorithm (which stated in Section 3) is $O(|E|)$. Hence, the time complexity of the VMtoDNAssignment algorithm is $O(|E|log(|E|) + |V|^2 \times (|V| + |E| + |E|))$. As $|E| = |V| \times |D|$ and $log(|E|) = log(|V| * |D|) \leq |V|^2$, the time complexity of the VMtoDNAssignment algorithm equals to $O(|V|^3 \times |D|)$.

It is worthwhile to mention that, however, the time complexity of the VMtoDNAssignment algorithm seems to be high for cloud networks, it is less than the time complexity of the existing algorithms, e.g., the proposed algorithm in Ref. [7]. Indeed, the VMtoDNAssignment algorithm runs in reasonable time in simulation experiments.

5 Capacity Adaptability

There are situations that a job which already has a VM assignment, needs more VMs to continue its execution properly; for example, when some new DNs are added or some DNs need more assigned VMs [13]. The adaptability of the allocated capacity for a job is related to the concept of elasticity which is one of important and outstanding characteristic of cloud systems. In this section, we present the ElasticityAssignment algorithm to handle this adaptability. We assume that the initial VM assignment for a job is calculated by the VMtoDNAssignment algorithm and the secondary changes for the VM assignment related to a job are handled by the ElasticityAssignment algorithm.

The general idea of the ElasticityAssignment algorithm is the same as the *VMtoDNAssignment* algorithm. Each pair of possible VMs is examined (as the diameter) to find a complementary VM assignment for the new added DNs. At last, the algorithm selects the best candidate VM assignment such that the maximum latency of the overall VM assignment (i.e., the complementary VM assignment in addition to the existing VM assignment) be minimum.

5.1 ElasticityAssignment Algorithm

The ElasticityAssignment algorithm handles the changes related to an existing job in the from of increasing the number of allocated VMs and assigning them to new DNs. This algorithm considers the current VM assignment of the job and assigns a subset of available VMs to unmatched DNs to minimize communication latency of the overall VM assignment. In the rest of this section, let V, V_{init}, D_{init}, D_{new} and MM be the set of available VMs, the set of assigned VMs, the initial set of DNs, the set of new added DNs and the matching from V_{init} to D_{init}, respectively.

The ElasticityAssignment algorithm searches for a set $V_{new} \subset V$ and a matching from D_{new} to V_{new}, such that the maximum latency of the set $V_{init} \cup V_{new}$ and their assignment to $D_{init} \cup D_{new}$ be minimum. Initially, the algorithm finds the diameter of the set V_{init} which its end nodes are denoted by dmr_1 and dmr_2. Then, the algorithm examines possible VM assignments for every pair $(v_i, v_j) \in (V \times V) \cup (V \times V_{init}) \cup \{(dmr_1, dmr_2)\}$. If the set $MaximalSet(v_i, v_j)$ contains the set V_{init}, then the perfect matching with minimum diam for the $MaximalSet(v_i, v_j)$ is calculated by calling the MinMatching algorithm. The latency of the overall assignment (i.e., V_{init} and a subset of $MaximalSet(v_i, v_j)$ with their assignment to $D_{init} \cup D_{new}$) is compared with the latency of the best found VM assignment. The best found candidate is updated if its latency is greater than the latency of this overall assignment. Finally, when all mentioned pairs are examined, the best found VM assignment is returned. The detail of the ElasticityAssignment algorithm is illustrated in Algorithm 5.

Algorithm 5. *ElasticityAssignment* Algorithm

Require: a set V_{init} (allocated VMs), a set D_{new} (new DNs), a set V (available VMs) and $\{d_{u,v} | (u,v) \in (D_{new} \cup V) \times (D_{new} \cup V)\}$
1: candidateSet ← a random VM assignment
2: $L \leftarrow$ sorted list $\{d_{u,v} | u \in D_{new}$ and $v \in V\}$
3: $(dmr_1, dmr_2) \leftarrow$ diameter of V_{init}
4: **for** each $(v_i, v_j) \in (V \times V) \cup (V \times V_{init}) \cup \{(dmr_1, dmr_2)\}$ **do**
5: $s_{max} \leftarrow$ MaximalSet(v_i, v_j)
6: **if** $V_{init} \not\subset s_{max}$ **then**
7: continue
8: **end if**
9: **if** $|s_{max}| \geq |D_{new}|$ **then**
10: $L_{D_{new},s_{max}} \leftarrow$ sorted list $\{d_{u,v} | u \in D_{new}$ and $v \in s_{max}\}$
11: $M \leftarrow$ MinMatching$(D_{new}, s_{max}, L_{D_{new},s_{max}})$
12: $A \leftarrow$ VM assignment related to M
13: **if** A is better than candidateSet **then**
14: candidateSet ← A
15: **end if**
16: **end if**
17: **end for**
18: **return** candidateSet

5.2 Optimality

In this section, a theorem is provided to show the optimality of the ElasticityAssignment algorithm.

Theorem 5.1. *The ElasticityAssignment algorithm finds a VM assignment from $V_{new} \in V$ to D_{new} such that the latency of the overall VM assignment $V_{new} \cup V_{init}$ to $D_{new} \cup D_{init}$ be minimum when the quadrilateral inequality is satisfied on latencies.*

Proof. We assume, by contradiction, that there is a VM assignment C with a latency less than the latency of the output of the ElasticityAssignment algorithm. In the assignment C, the set of new assigned VMs and the corresponding matching are V_c and MM_c, respectively. Also, the diameter of the set V_c is (di_{1c}, di_{2c}).

Let consider the VM assignment $\overline{C}$ that is concluded from the examination of the pair (di_{1c}, di_{2c}) in Line 4 of the ElasticityAssignment algorithm. The VM assignment $\overline{C}$ exists because s_{max} is a superset of the set V_c and the If statement in Line 5 is not satisfied. Also, let consider the VM assignment $\overline{C}$ included the set $V_{\overline{c}}$ of VMs and the matching $MM_{\overline{c}}$.

Here, the latency of the assignment $\overline{C}$ in addition to the initial VM assignment is verified. This latency is the maximum of the latency $V_{\overline{c}} \cup V_{init}$ and the latency $MM_{\overline{c}} \cup MM$. As $V_{\overline{c}} \subset MaximalSet(di_{1c}, di_{2c})$ and $V_{init} \subset MaximalSet(di_{1c}, di_{2c})$ (Line 6 of the ElasticityAssignment algorithm), the latency $V_{\overline{c}} \cup V_{init}$ equals to $d_{di_{1c},di_{2c}}$. Also, the latency of the matching $MM_{\overline{c}}$ is not greater than the latency of the matching MM_c because the set $V_{\overline{c}}$ is selected regarding a matching from the set D_{new} to the set $MaximalSet(di_{1c}, di_{2c})$ while the matching MM_c is a possible candidate for it. Therefore, the latency of $MM_c \cup MM$ is not less than the latency of $MM_{\overline{c}} \cup MM$.

Overall, it is concluded that the candidate set from checking (di_{1c}, di_{2c}) in Line 4 of the ElasticityAssignment algorithm does not have greater latency in comparison to the VM assignment C.

The maximum latency can be one of the following cases (which are verified in Line 4 of the ElasticityAssignment algorithm) when the VM assignment C is considered in addition to the initial VM assignment. Note that in all the cases the diameter of the set $V_{init} \cup V_c$ is examined by the ElasticityAssignment algorithm:

- An edge in V_{init} that has to be the same as the diameter of the set V_{init} and is verified as $\{(dmr_1, dmr_2)\}$.
- An edge in V_c that is verified as an edge in $V \times V$ because $V_c \subset V$,.
- A member in $V_{init} \times V_c$ that is verified as an edge in $V \times V_{init}$.
- In the case of an edge in MM_c or MM, the diameter of $V_{init} \cup V_c$ is one of the three above cases.

Overall, the existence of the VM assignment C is a contradiction. That is because the VM assignment $\overline{C}$ is not worse than the VM assignment C while the output of the ElasticityAssignment algorithm is not worse than the VM assignment $\overline{C}$. □

5.3 Time Complexity

The discussion of the time complexity of the ElasticityAssignment algorithm is similar to the VMtoDNAssignment algorithm. The diameter of the set V_{init} in Line 3 is found in $O(|V_{init}|^2)$. Further, the number of iterations of the main loop is $O(|V|^2)$. Each iteration has the time complexity $O(|V| + |E|)$ regarding the MaximalSet algorithm and the MinMatching algorithm. Hence, the ElasticityAssignment algorithm is executed in $O(|V|^3|D|)$.

Note that it is possible to remove heuristically some nodes from the set V to reduce the run time. For example, after the execution of the VMtoDNAssignment algorithm, there are VMs that are far from the set V_{init} and can be flagged not to consider in the executions of the ElasticityAssignment algorithm for a particular job.

6 Evaluation

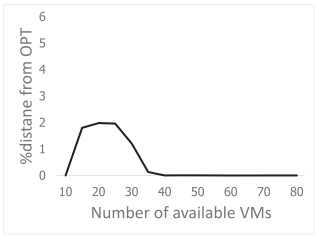
In this section, we conduct experiments to evaluate the performance of the VMtoDNAssignment algorithm. We assume a data center setup with 1,024 racks. We consider five topologies inside data centers. These topologies are Hierarchical, Tree, VL2, Fat-tree and BCube. The properties of these topologies are discussed in references [6], [7], [11]. The communication latency between two nodes (in these topologies) is related to the number of switches between them while the effect of link latency is considered in the number of switches. The number of switches between two nodes in rack intervals {[1-16], [1-64], [1-256], [1-1024]} are {1, 3, 5, 7} in Hierarchical topology, {1, 3, 3, 5} in Tree topology, {1, 5, 5, 5} in VL2 topology, {3, 3, 5, 5} in Fat-tree topology and {1, 3, 3, 3} in BCube topology.

In Section 6.1, we provide two scenarios to evaluate the performance and scalability of the VMtoDNAssignment algorithm regarding 1) the number of VMs and 2) the relation between latency and the number of switches (in a communication path). In Section 6.2, we compare the VMtoDNAssignment algorithm with the rVMPDN algorithm [7] which is an algorithm for the considered problem. The comparison is based on the proposed scenario in Ref. [7]. In Section 6.3, a scenario is presented to evaluate the performance of the ElasticityAssignment algorithm with regard to the different number of VMs. Finally, in Section 6.4, we provide a study on the performance variation of our proposed algorithms in the considered scenarios.

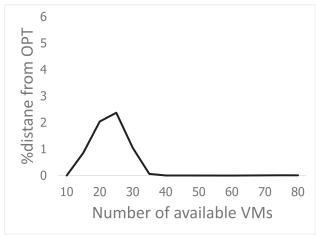
In all scenarios, we compare the output latency of our proposed algorithm (which denoted by our latency) with the optimal latency (which denoted by OPT). Note that OPT can be obtained by examining all possible VM assignments. Each comparison result is reported as the difference between our latency and OPT that is expressed as a percentage relative to OPT. We run 100 experiments for each case and the average and maximum of its standard deviation are reported.

6.1 Performance Evaluation of the VMtoDNAssignment Algorithm

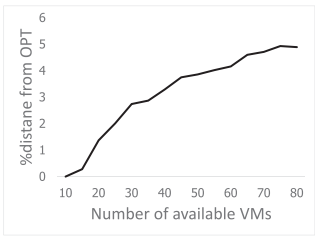
In the first scenario, we consider the number of DNs is 10 and the number of available VMs takes different values from 10 to 80 (by steps 5). The placement of VMs and DNs are selected randomly from the set of 1,024 racks (each rack is a node that can be one VM or one DN). The latency between two racks is taken as a random number from interval $[0.9 - 1.1]$ times the number of switches between them [14]. This scenario is repeated for each topology that the results are reported in Fig. 4. The results show that the distance of our latency from OPT is reduced by increasing the number of VMs in Tree and Hierarchical topologies (from 25 VMs in this scenario). Also, the diagram of the mentioned distance for other topologies have a smaller slope for larger number of VMs. In Section 6.4, we discuss the reason that the performance of our proposed algorithms varies according to the number of VMs.



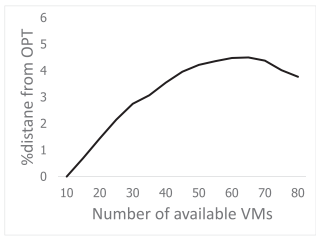
(a) Hierarchical



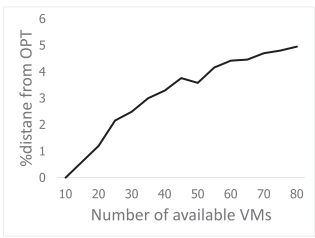
(b) Tree



(c) VL2



(d) Fat-tree



(e) BCube

	AVG(SD)	MAX(SD)
Hierarchical	0.4	1.6
Tree	0.42	1.92
VL2	0.75	0.91
Fat-tree	1.18	2.64
BCube	0.76	0.96

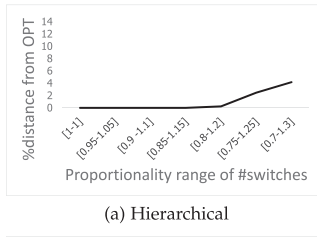
(f) Standard Deviation in percentage

Fig. 4. The distance of our latency from OPT (the optimal latency) for different numbers of available VMs.

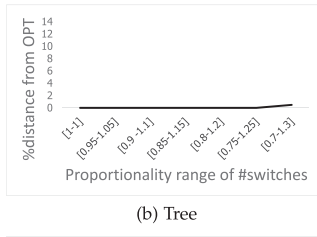
In the second scenario, we consider the number of DNs and the number of available VMs are 10 and 60, respectively. The placements of VMs and DNs are selected randomly from the set of 1,024 racks. The latency between two racks is taken as a random number from interval $[r_1 - r_2]$ times the number of switches between them. The interval $[r_1 - r_2]$ has seven following cases: $[1.0 - 1.0]$, $[0.95 - 1.05]$, $[0.9 - 1.1]$, $[0.85 - 1.15]$, $[0.8 - 1.2]$, $[0.75 - 1.25]$ and $[0.7 - 1.3]$. The results of this scenario are reported in Fig. 5. These results show that the performance of the VMtoDNAssignment algorithm is related linearly to the proportion of latency to the number of switches, i.e., it works better when the latencies of different switches (and different links) differ less. More difference between latency of various switches is model by larger intervals. The mentioned intervals indicate how much the considered latencies can be differ from the latencies of a network that the quadrilateral inequality holds (for interval [1.0-1.0], the quadrilateral inequality holds and shorter intervals lead to better estimation of the quadrilateral inequality).

6.2 Comparisons with an Existing Algorithm

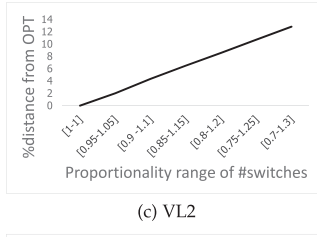
In this section, we compare the VMtoDNAssignment algorithm with the rVMPDN algorithm by conducting the simulation as described in Ref. [7]. In this scenario, there are 120 and 40 VMs and DNs, respectively. The placements of nodes are selected randomly from the set of first 1,024 racks, 256 racks, 64 racks and 16 racks while the latency between two racks is taken as a random number from interval $[0.9 - 1.1]$ times the number of switches between them. The worst ratios of the maximum access latency of the two considered algorithms with regard to different topologies are illustrated in Fig. 6. Note that the ratios related to the



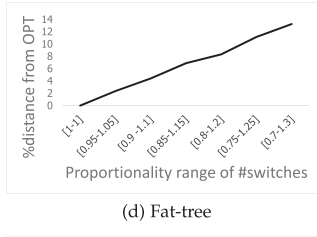
(a) Hierarchical



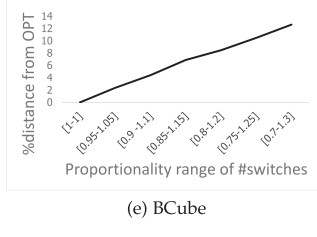
(b) Tree



(c) VL2



(d) Fat-tree



(e) BCube

	AVG(SD)	MAX(SD)
Hierarchical	0.67	2.47
Tree	0.12	0.85
VL2	1.28	2.49
Fat-tree	1.94	4.07
BCube	2.35	4.07

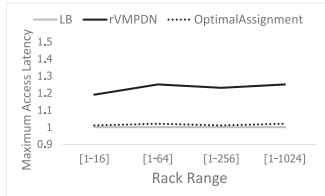
(f) Standard Deviation in percentage

Fig. 5. The distance of our latency from OPT (the optimal latency) for various proportions of latency to the number of switches.

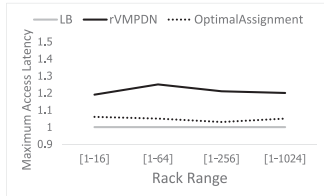
rVMPDN algorithm is taken from Ref. [7]. The worst ratio of the maximum access latency of the VMtoDNAssignment algorithm to LB (lower bond) ranges from 1.01 to 1.06 while this ratio of the rVMPDN algorithm to LB ranges from 1.17 to 1.22. Also, our simulation results show that the run time of the VMtoDNAssignment algorithm (like the time complexity) is lower than the run time of the rVMPDN algorithm. Overall, the VMtoDNAssignment algorithm runs faster and finds a VM assignment with smaller latency in comparisons to the rVMPDN algorithm.

6.3 Performance Evaluation of the Elasticity Assignment Algorithm

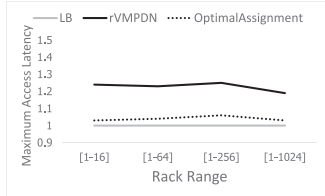
In this scenario, the performance of the ElasticityAssignment algorithm is evaluated. The number of available VMs takes different values from 10 to 80 (by steps 5). Also, the



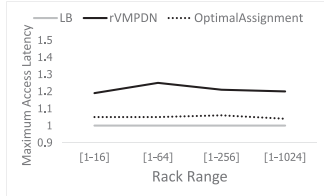
(a) Tree



(b) VL2

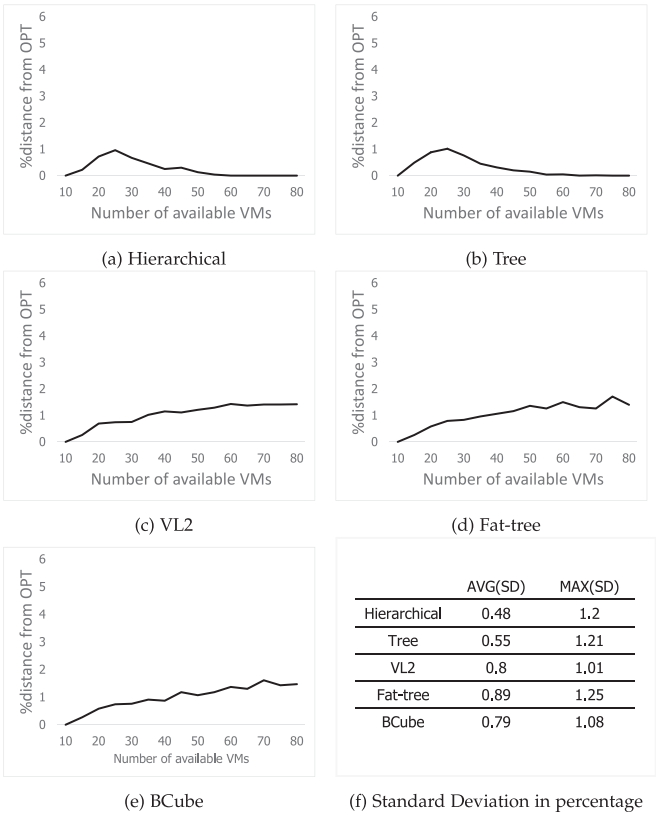


(c) Fat-Tree



(d) BCube

Fig. 6. The worst ratios of the maximum access latencies of the VMtoDNAssignment algorithm and the rVMPDN algorithm.



	AVG(SD)	MAX(SD)
Hierarchical	0.48	1.2
Tree	0.55	1.21
VL2	0.8	1.01
Fat-tree	0.89	1.25
BCube	0.79	1.08

(f) Standard Deviation in percentage

Fig. 7. The impact of the number of VMs in the ElasticityAssignment algorithm.

placements of VMs and DNs are selected randomly from the set of 1,024 racks. This scenario is repeated for each topology while the latency between two racks is taken as a random number from interval $[0.9 - 1.1]$ times the number of switches between them. We consider the initial number of DNs is 7; then 3 new DNs are added to the set of DNs. The initial assignment of VMs to 7 DNs is calculated by the VMtoDNAssignment algorithm. Then the assignment of 3 new added DNs is calculated by the ElasticityAssignment algorithm. The result of this scenario is illustrated in Fig. 7. The results show that the performance and the scalability of the ElasticityAssignment algorithm are similar to the VMtoDNAssignment algorithm. In Section 6.4, we discuss the reason that the performance of our proposed algorithms varies according to the number of VMs.

6.4 On the Performance of the Proposed Algorithms

In this section, we discuss the performance variation of our proposed algorithms which is limited by the approximation of the quadrilateral inequality. Note that our proposed algorithms find the optimal latency when the quadrilateral inequality holds in a network. In what follows, initially, we state when the quadrilateral inequality can be approximated well in a generated network. Subsequently, we examine the randomly generated networks of different topologies. Finally, we conclude this section by discussing the performance variation of our proposed algorithms in Figs. 4 and 7.

In our simulation, we select a subset of nodes (as VMs) from the set of racks and randomly select the distances between them (regarding the number of switches between them) to generate the communication graph. If the values of the number of switches between any two selected nodes are the same, then the generated graph can be any random graph. Therefore, the quadrilateral inequality is not approximated well in the generated network, necessarily. For example, if 16 nodes are selected and all of them be in a same rack interval $\#16$, then there is one switch between any two nodes and the assumed distances between nodes are random numbers from interval $[0.9 - 1.1]$.

Let assume a generated graph that the values of the number of switches between selected racks can be one, three and five. The generated graph is not any general graph as assumed distances related to the case of one switch, three switches and five switches are distinguishable from each other. The quadrilateral inequality can be approximated better in such graphs in comparison to a general graph. That is because it is possible to sort the distances and assume the quadrilateral inequality between them. Let consider two cases for selecting 16 nodes. First case, 16 nodes are in a rack interval $\#16$, therefore, there is one switch between any two nodes. There is low probability that the quadrilateral inequality can be approximated well in the generated graph. Second case, 16 nodes are in different rack intervals $\#64$, therefore, the values of the number of switches between nodes are various and the quadrilateral inequality can be approximated well. Overall, with high probability, the quadrilateral inequality can be approximated better in the first case than the second case.

Let assume Hierarchical topology. If there are less than 17 nodes, then they can be in one rack interval $\#16$ such that there is one switch between any two nodes. But, if there are more than 16 nodes, then they can not be in one rack interval $\#16$, therefore, all the distances are not related to the case of one switch. Also, if there are more than four nodes, then there are two nodes in one rack interval $\#16$ or in two different rack intervals $\#64$, therefore, all the distances are not related to the case of only three switches. The case of only five switches is similar to the case of only three switches. Overall, when the number of selected nodes are more than 16, the values of the number of switches between nodes are not the same (e.g., if 256 nodes in Hierarchical topology are selected, then the values of the number of switches between nodes can be one, three and five).

In Hierarchical and Tree topologies, when the number of VMs is less than or equal to 16, there are a lot of possible generated graphs that the quadrilateral inequality can not be approximated well in them. Hence, when the number of VMs is in interval $[10 - 16]$, the distance from OPT is more for larger number of VMs. By increasing the number of VMs from 16, there are more generated graphs that the quadrilateral inequality can be approximated well in them, therefore, the distance from OPT starts to diminish.

In VL2 and BCube topologies, if one node is selected from each rack interval $\#16$ (i.e., 64 nodes from different rack intervals $\#16$), then the number of switches between any two nodes is the same (three for BCube and five for VL2) and any general graph can be generated. For Fat-tree topology, if 64 nodes are selected from a rack interval $\#64$, then the number of switches between any two nodes is three. In these topologies, increasing the number of selected VMs (from 10 to 64) leads to more generated networks

which the quadrilateral inequality can not be approximated well. Hence, the distance from OPT is more for larger number of VMs when the number of selected nodes is in interval $[10-64]$. However, by increasing the number of VMs from 64, the values of the number of switches between selected racks are various and the quadrilateral inequality can be approximated better for larger number of VMs.

Overall, in Figs. 4 and 7, the reason of the performance variation is that the number of selected nodes determines the probability of a good approximation of the quadrilateral inequality. When the number of VMs is low (which its threshold is different for the considered topologies), the selected nodes can be in a rack interval (that there is the same number of switches between any two nodes), hence, the random generation does not benefit from the quadrilateral inequality in initial topology of racks. However, When the number of VMs is high (which its threshold is different for the considered topologies), the links between nodes have different numbers of switches, therefore, randomly generated distances are distinguishable and the quadrilateral inequality approximation benefits from it.

In the considered topologies, by increasing the number of VMs from 10, the performance of the proposed algorithms first decreases and then increases. For example, in Fig. 4b the performance decreases in interval $[10-25]$ and increases in interval $[25-40]$. However, all of the performance changes are not shown in Figs. 4 and 7 as our simulation is limited to 80 VMs. For example, the peak of performance is not shown in Fig. 4e. In each diagram, the maximum standard deviation is related to the peak of diagram. Hence, the maximum standard deviations related to VL2 and BCube topologies are less than the others as their peaks are not in the diagrams. However, average standard deviations of Hierarchical and Tree topologies are less than the others. That is because when there are more than 40 VMs, the standard deviation is approximately zero. These zeros reduce the average of the standard deviation.

7 Related Work

More recently, VM placement in cloud data centers has been addressed in various aspects [5], [15], [16]. Some of the important aspects are energy [17], [18], [19], network [20], [21], [22] and service reliability [23], [24]. Network-aware VM allocation minimizing the latency of the communication and data access needs intelligent VM placement. For example, in applications like MapReduce, VMs are assigned to DNs while the communication latencies are important and have to be bounded [25], [26], [27], [28]. The problem of VM placement with respect to the latency is a combination of the resource allocation and the assignment problem (that is, a subset of VMs are allocated to a request by considering their assignment to DNs). Further, the considered problem can be applied to the cloud elasticity as handling future changes in a VM request.

The studies on the network-aware VM allocation are widely available in literature like [11], [29], [30], [31], [32], [33]. More specifically, the communication latency of a VM assignment has been considered in literature like [6], [7], [34], [35], [36], [37]. Several authors examined the VM placement under the assumption that the triangle inequality holds in the communication latencies. Alicherry et al. [36] gave the first version of the problem to minimize the network diameter of a VM allocation. They showed that their problem is NP-hard and provided a 2-approximation algorithm for it. As an extension of VM allocation to consider the data access, Alicherry et al. [6] defined the problem of the assignment of VMs to DNs to minimize the maximum latency between VMs and the latency between DNs to their assigned VMs. Kuo et al. [7] showed that this problem with the triangle inequality assumption does not have a solution better than 2-approximation and provided a 2-approximation algorithm.

In this paper, we define a new property for communication latency such that, in contrast to mentioned studies, our problem definition is not limited to the triangle inequality. Our defined property holds in tree topologies and is well estimated for the cloud networks. Similarly, Ref. [37] is not limited to the assumption of the triangle inequality such that it presented an algorithm to find a latency optimal VM allocation in tree networks (not include data access).

The sub problem of assigning selected VMs to considered DNs can be considered as a perfect matching with minimum diam (the edge with maximum weight) in unbalanced weighted bipartite graph. The Hopcroft-Karp algorithm produces as output a maximum cardinality matching for a bipartite graph. The mentioned matching is obtained by calling this algorithm in binary fashion on the sorted list of edges. However, the time complexity can be improved. Besides, the maximum/minimum perfect matching in unbalanced weighted bipartite graph is defined as selecting a subset of edges such that each node in smaller part has exactly one match in larger part while the total weight of selected edges is maximum/minimum [38]. In this definition, the total weight of edges is considered while in our problem the edge with maximum weight is important. The best solution for maximum/minimum perfect matching in unbalanced weighted bipartite graph was provided in Ref. [12], however, it is not applicable for our problem due to the different definition of the optimal matching. Overall, in this paper, we present an algorithm (to find a perfect matching with minimum diam in unbalanced weighted bipartite graph) that its time complexity is better than the time complexity of the existing methods.

Another important aspect of the VM placement problem (regarding the dynamic environment of cloud systems) is the support of future changes in job requirements or job resources. Elastic computing is a concept in cloud computing in which computing resources have to be scaled up and down easily by the cloud service provider [9]. The elasticity of resources can be in the form of different items like processing power, storage and bandwidth. The elasticity in VM placement has been received attention in the literature [39], [40]. For example, Ref. [41] proposed an elasticity-aware hierarchical VM placement algorithm. However, considering the literature, the VM placement needs more attention regarding elasticity. Our method can be applied to increase the number of allocated VMs and provides a complementary VM assignment with overall good latency while there are not similar studies in the literature.

8 Conclusion and Future Work

Prior studies on the problem of assigning VMs to DNs, assumed the triangle inequality on the communication and access latencies while they showed that there is no approximation better than two for this problem. However, the assumption of the triangle inequality can be substituted by a more suitable one. Here, we define the quadrilateral inequality property which applies to tree and hierarchical topologies. The problem of VM placement with the quadrilateral inequality is P. In this paper, an algorithm is proposed to find an assignment of VMs to DNs with minimum communication latency while the hardness of the problem is changed by considering the quadrilateral inequality rather than the triangle inequality. For example, this algorithm can be used in application like MapReduce. The theoretical proof shows that the proposed algorithm finds minimum latency in topologies like tree and hierarchical. Further, the experimental results validate the scalability and performance of the proposed algorithm for several known data center networks. Therefore, this algorithm can be used in the scale of cloud systems.

Overall, the novelty of this paper is fourfold. First, we revise the assumption of the triangle inequality by the quadrilateral inequality. Second, we provide an algorithm to find a perfect matching with minimum diam in unbalanced weighted bipartite graphs. Third, we present a method to return an assignment of VMs to DNs with minimum latency. Fourth, we extend our method to be utilized for increasing the number of requested VMs (elasticity). Further, as a future work, one can present an algorithm to serve several requests simultaneously and consolidate the allocated VMs.

References

[1] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of big data on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, 2015.

[2] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[3] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (IaaS) in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 41, pp. 424–440, 2014.

[4] 2015. [Online]. Available: http://hadoop.apache.org/

[5] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *J. Netw. Syst. Manage.*, Springer, vol. 23, pp. 1–53, 2014.

[6] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proc. IEEE INFOCOM*, 2013, pp. 647–655.

[7] J. Kuo, H. Yang, and M. Tsai, "Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems," in *Proc. IEEE INFOCOM*, 2014, pp. 1303–1311.

[8] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *Data Center Networks - Topologies, Architectures and Fault-Tolerance Characteristics*. Berlin, Germany: Springer, 2013.

[9] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proc. 10th Int. Conf. Autonomic Comput.*, 2013, pp. 23–27.

[10] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, "Elasticity in cloud computing: A survey," *Annales des Télécommunications*, vol. 70, no. 7/8, pp. 289–309, 2015.

[11] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010, pp. 1154–1162.

[12] L. Ramshaw and R. E. Tarjan, "On minimum-cost assignments in unbalanced bipartite graphs," HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1, 2012.

[13] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Proc. Int. Conf. Collaboration Technol. Syst.*, 2013, pp. 42–47.

[14] C. Yu, C. Lumezanu, A. B. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *Proc. 16th Int. Conf. Passive Active Meas.*, 2015, pp. 360–372.

[15] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (IaaS) in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 41, pp. 424–440, 2014.

[16] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *J. Netw. Comput. Appl.*, vol. 66, no. C, pp. 106–127, May 2016.

[17] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surveys*, vol. 47, no. 2, pp. 33:1–33:36, Dec. 2014.

[18] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM Int. Conf. Cluster Cloud Grid Comput.*, 2010, pp. 826–831.

[19] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 755–768, May 2012.

[20] M. A. Sharkh, M. Jammal, A. Shami, and A. H. Ouda, "Resource allocation in a network-based cloud computing environment: Design challenges," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 46–52, Nov. 2013.

[21] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: Taxonomy and survey," *Softw. Practice Experience*, vol. 44, no. 3, pp. 369–390, 2014.

[22] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proc. IEEE*, vol. 102, no. 1, pp. 11–31, Jan. 2014.

[23] A. Zhou, S. Wang, Z. Zheng, C. H. Hsu, M. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 452–466, Oct.–Dec. 2016.

[24] J. Liu, S. Wang, A. Zhou, S. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, p. 1, 2016.

[25] M. S. Yoon and A. E. Kamal, "Optimal dataset allocation in distributed heterogeneous clouds," in *Proc. IEEE Globecom Workshops*, 2014, pp. 75–80.

[26] T. Gunarathne, B. Zhang, T. Wu, and J. Qiu, "Scalable parallel computing on clouds using Twister4Azure iterative MapReduce," *Future Generation Comput. Syst.*, vol. 29, no. 4, pp. 1035–1048, 2013.

[27] L. Wang, et al., "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," *Future Generation Comput. Syst.*, vol. 29, no. 3, pp. 739–750, 2013.

[28] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware resource allocation for MapReduce in a cloud," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, 2011, pp. 58:1–58:11.

[29] J. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Proc. 9th Int. Conf. Grid Cloud Comput.*, 2010, pp. 87–92.

[30] D. Chang, G. Xu, L. Hu, and K. Yang, "A network-aware virtual machine placement algorithm in mobile cloud computing environment," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops*, 2013, pp. 117–122.

[31] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, and A. Wolman, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. 7th USENIX Symp. Netw. Syst. Des. Implementation*, 2010, pp. 17–32.

[32] G. Luo, Z. Qian, M. Dong, K. Ota, and S. Lu, "Network-aware rescheduling: Towards improving network performance of virtual machines in a data center," in *Proc. 14th Int. Conf. Algorithms Archit. Parallel Process.*, 2014, pp. 255–269.

[33] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Network-aware virtual machine placement and migration in cloud data centers," in *Emerging Research in Cloud Distributed Computing Systems*. Hershey, PA, USA: IGI Global, 2015, pp. 42–91.

[34] E. Casalicchio, D. A. Menascé, and A. Aldhalaan, "Autonomic resource provisioning in cloud systems with availability goals," in *Proc. ACM Cloud Autonomic Comput. Conf.*, 2013, pp. 1:1–1:10.

[35] J. A. Pascual, T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Towards a greener cloud infrastructure management using optimized placement policies," *J. Grid Comput.*, Springer, vol. 13, no. 3, pp. 375–389, 2015.
[36] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. IEEE INFOCOM*, 2012, pp. 963–971.
[37] M. Malekimajd, A. Movaghar, and S. Hosseinimotlagh, "Minimizing latency in geo-distributed clouds," *J. Supercomputing*, vol. 71, no. 12, pp. 4423–4445, 2015.
[38] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput.*, vol. 2, no. 4, pp. 225–231, 1973.
[39] S. He, L. Guo, Y. Guo, C. Wu, M. Ghanem, and R. Han, "Elastic application container: A lightweight approach for cloud resource provisioning," in *Proc. IEEE 26th Int. Conf. Adv. Inf. Netw. Appl.*, 2012, pp. 15–22.
[40] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Elastic management of web server clusters on distributed virtual infrastructures," *Concurrency Comput.: Practice Experience*, vol. 23, no. 13, pp. 1474–1490, 2011.
[41] K. Li, J. Wu, and A. Blaisse, "Elasticity-aware virtual machine placement for cloud datacenters," in *Proc. IEEE 2nd Int. Conf. Cloud Netw.*, 2013, pp. 99–107.

Marzieh Malekimajd received the BSc and MSc degrees in computer engineering from the Sharif University of Technology, in 2009 and 2011, respectively. She was a visiting PhD student in the Department of Electronics, Information, and Bioengineering, Politecnico di Milano, in 2014. She is currently working toward the PhD degree in computer engineering at the Sharif University of Technology. Her research interests include graph theory and algorithmic aspects of cloud computing, MapReduce, and networks.

Ali Movagahr received the BS degree in electrical engineering from the University of Tehran, in 1977, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He is a professor in the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, and has been on the Sharif faculty since 1993. He visited the Institut National de Recherche en Informatique et en Automatique, Paris, France, and the Department of Electrical Engineering and Computer Science, University of California, Irvine, in 1984 and 2011, respectively, worked at AT&T Information Systems in Naperville, Illinois in 1985-1986, and taught at the University of Michigan, Ann Arbor in 1987-1989. His research interests include performance/dependability modeling and formal verification of wireless networks, distributed real-time systems and cyber-physical systems. He is a senior member of the IEEE and the ACM.

Minimizing Redundancy to Satisfy Reliability Requirement for a Parallel Application on Heterogeneous Service-Oriented Systems

Guoqi Xie, *Member, IEEE*, Gang Zeng, *Member, IEEE*, Yuekun Chen, Yang Bai, Zhili Zhou, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—Reliability is widely identified as an increasingly relevant issue in heterogeneous service-oriented systems because processor failure affects the quality of service to users. Replication-based fault-tolerance is a common approach to satisfy application's reliability requirement. This study solves the problem of minimizing redundancy to satisfy reliability requirement for a directed acyclic graph (DAG)-based parallel application on heterogeneous service-oriented systems. We first propose the enough replication for redundancy minimization (ERRM) algorithm to satisfy application's reliability requirement, and then propose heuristic replication for redundancy minimization (HRRM) to satisfy application's reliability requirement with low time complexity. Experimental results on real and randomly generated parallel applications at different scales, parallelism, and heterogeneity verify that ERRM can generate least redundancy followed by HRRM, and the state-of-the-art MaxRe and RR algorithm. In addition, HRRM implements approximate minimum redundancy with a short computation time.

Index Terms—Fault-tolerance, heterogeneous service-oriented systems, quality of service, reliability requirement, replication

1 Introduction

1.1 Background

Cloud-based service is a new service-based resource sharing paradigm [1], [2]. In X as a service (XaaS) clouds, resources as services (e.g., infrastructure, platform and software as a service) are sold to applications such as scientific and big data analysis workflows [1], [3], [4], [5], [6]. Meanwhile, cloud computing systems become more heterogeneous as old, slow machines are continuously replaced with new, fast ones. Heterogeneous computing systems consist of diverse sets of processors interconnected with a high-speed network, and are applied in business-critical, mission-critical, and safety-critical scenarios to achieve operational goals [7]. Applications in the system are increasingly parallel and the tasks in an application have obvious data dependencies and precedence constraints [1], [8], [9], [10], [11]. Examples of parallel applications are Gaussian elimination and fast Fourier transform [9]. A parallel application with precedence constrained tasks at a high level is described by a directed acyclic graph (DAG) [1], [8], [9], [10], [11], where nodes represent tasks, and edges represent communication messages between tasks. Such application is usually called DAG-based parallel application [12].

The current cloud-based service systems are actually heterogeneous service-oriented systems where resource management is a considerable challenge owing to the various configurations or capacities of the hardware or software [13]. The processing capacity of processors in heterogeneous service-oriented systems has been developed to provide powerful cloud-based services, whereas failures of processors will affect the reliability of systems and quality of service (QoS) for users [2]. Reliability is defined as the probability of a schedule successfully completing its execution, and it has been widely identified as an increasingly relevant issue in service-oriented computing systems [2], [14], [15], [16], [17], [18].

Fault-tolerance by primary-backup replication, which means that a primary task will have zero, one, or multiple backup tasks, is an important reliability enhancement mechanism. In the primary-backup replication scheme, the primary and all the backups are called replicas. Although replication-based fault-tolerance is an important reliability enhancement mechanism [14], [15], [19], [20], [21], any application cannot be 100 percent reliable in practice. Therefore, if an application can satisfy its specified reliability requirement (also named reliability goal or reliability assurance in some studies), then it is considered to be reliable. For example, assume that the application's reliability requirement is 0.9, only if the application's reliability exceeds 0.9, will the application be reliable. Specifically,

- *G. Xie, Y. Chen, Y. Bai, and R. Li are with the College of Computer Science and Electronic Engineering, Hunan University, Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan 410082, China. E-mail: {xgqman, baiyang, lirenfa}@hnu.edu.cn, chenyuekun@126.com.*
- *G. Zeng is with the Graduate School of Engineering, Nagoya University, Aichi 4648603, Japan. E-mail: sogo@ertl.jp.*
- *Z. Zhou is with the Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: zhou_zhili@163.com.*
- *K. Li is with the College of Computer Science and Electronic Engineering, Hunan University, Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan 410082, China, and the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 14 Aug. 2016; revised 1 Jan. 2017; accepted 3 Feb. 2017. Date of publication 7 Feb. 2017; date of current version 13 Oct. 2020. Digital Object Identifier no. 10.1109/TSC.2017.2665552

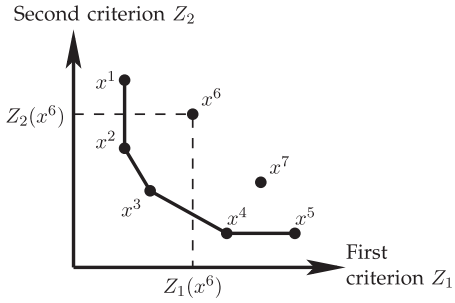


Fig. 1. Pareto optima and pareto curve for a bicriteria minimization problem [19].

reliability requirement has been defined in some reliability related standards (e.g., IEC 61508 [22] and ISO 9000 [23]), and it is one of the most important QoS in cloud and services computing systems [14], [15]. Therefore, reliability requirement must be satisfied from standards and QoS perspectives. However, as pointed out in [2], many cloud-based services failed to fulfill their reliability requirements due to processor failures in practice.

1.2 Motivation

Users and resource providers are the two types of roles with different requirements for service-oriented systems [24]. For users, satisfying application's reliability requirement is one of the most important QoS requirements, for which replication-based fault-tolerance is a common approach. For resource providers, minimizing resource redundancy caused by replication is one of the most important concerns [14], [15]. However, adding more replicas (including primary and backups) could increase both reliability and redundancy for a parallel application. Therefore, both criteria (low redundancy and high reliability, short schedule length and high reliability) are conflicting, and optimizing them is a bi-criteria optima problem [19]. In Fig. 1, each point x^1-x^7 represents a solution of a bicriteria minimization problem [19]. The points x^1, x^2, x^3, x^4, and x^5 are Pareto optima [25]; the points x^1 and x^5 are weak optima, whereas the points x^2, x^3, and x^4 are strong optima. The set of all Pareto optima is called the Pareto curve [19]. Many studies have dealt specifically with the bi-criteria (i.e., minimizing schedule length and maximizing reliability) problem to obtain such an approximate Pareto curve for a DAG-based parallel application [19], [20], [21], [26], [27], [28]. In [26], [27], [28], the approaches increase reliability by efficient task scheduling without using replication. In [19], [20], [21], the approaches presented replicate tasks to increase reliability.

However, for heterogeneous service-oriented systems, resolving the above bi-criteria is not strictly required for the following reasons:

(1) Clouds allow flexible and dynamic resource allocations based on a pay-as-you-go scheme [29], where users pay only for the reliability requirement they apply and will not pay additional fees for the reliability that surpasses their reliability requirement.
(2) The application cannot be 100 percent reliable as mentioned earlier. The most common component of service-level agreement (SLA) between resource providers and the users is that the services (reliability requirement in this study) should be provided to the users as agreed upon in the contract [30]. Therefore, satisfying application's reliability requirement is the service level objective.

In summary, considering the actual demand, the theoretical bi-objective optimization problem could be degradated to a constrained single-objective optimization problem in most cases. In other words, reliability is not the higher the better, but as long as you can satisfy the reliability requirement from a practical perspective. Therefore, the reliability problem of service-oriented systems is mainly to satisfy application's reliability requirement while still reducing the resource as far as possible.

The approaches related to our work are [14] and [15], in which the authors presented the MaxRe and RR algorithms to minimize redundancy of a parallel application to satisfy application's reliability requirement on heterogeneous distributed systems. The main procedures of the MaxRe and RR are follows:

(1) The reliability requirement of the application is transferred to the sub-reliability requirements of the tasks. In this way, as long as the sub-reliability requirement of each task can be satisfied, the application's reliability requirement can be satisfied, such that a heuristic replication can be used in the following.
(2) MaxRe and RR iteratively assign the replicas of each task to the processors with maximum reliability values until the sub-reliability requirement of the task is satisfied.

However, the essential limitation of MaxRe and RR is that the sub-reliability requirements of tasks are too high, thereby causing them need unnecessary redundancy to satisfy the sub-reliability requirements.

1.3 Our Contributions

Similar to the state-of-the-art MaxRe and RR, this study aims to implement redundancy minimization to satisfy application's reliability requirement for a parallel application on heterogeneous service-oriented distributed systems. Our contributions comparing to the MaxRe and RR are summarized as follows:

(1) We present the just enough replication for redundancy minimization (ERRM) algorithm to satisfy application's reliability requirement by two-stage replications. The first stage involves obtaining the lower bound on redundancy (i.e., the minimum required number of replicas) for each task; the second stage is iteratively selecting the available replicas and corresponding processors with the maximum reliability values until application's reliability requirement is satisfied.
(2) To overcome the high time complexity of ERRM algorithm, we propose the heuristic replication for redundancy minimization (HRRM) algorithm to deal with large-scale parallel applications. Similar to the MaxRe and RR algorithms, HRRM first transfers the reliability requirement of the application to the sub-reliability requirements of the tasks. Then, HRRM iteratively assign the replicas of each task to the processors with maximum reliability values until

the sub-reliability requirement of the task is satisfied. The main improvement of HRRM over MaxRe and RR is that it can obtain lower sub-reliability requirements for most tasks, such that HRRM generates less redundancy than MaxRe and RR.

(3) Experimental results on real and randomly generated parallel applications at different scales, parallelism degrees, and heterogeneity degrees validate that ERRM can generate the least redundancy followed by HRRM, the state-of-the-art MaxRe and RR algorithm. In addition, HRRM implements approximate minimum redundancy with a short computation time.

The rest of this paper is organized as follows. Section 2 reviews related research. Section 3 presents the reliability modeling and problem statement. Section 4 explains the state-of-the-art MaxRe and RR algorithms. Sections 5 and 6 proposed the ERRM and HRRM algorithms, respectively. Section 7 verifies the ERRM and HRRM algorithms. Section 8 concludes this study.

2 Related Work

The widely-accepted reliability model was presented by Shatz and Wang [31], where each hardware component (processor) is characterized by a constant failure rate per time unit λ and the reliability during the interval of time t is $e^{-\lambda t}$. That is, the failure occurrence follows a constant parameter Poisson law [31]. This law is also known as the exponential distribution model [19]. This section mainly reviews the related research on reliability and fault-tolerance of DAG-based parallel applications.

Two main types of primary-backup replication approaches exist in current: active replication [14], [15], [20], [21] and passive replication [32], [33], [34], [35]. For the active replication scheme, each task is simultaneously replicated on several processors, and the task will succeed if at least one of them does not fail. For the passive scheme, whenever a processor fails, the task will be rescheduled to proceed on a backup processor. When a processor crashes, it is subsequently restarted to continue from the checkpoint just as if no failure had occurred; such scheme is called checkpoint and restart scheme, and can be considered as an improved version of the passive scheme [14], [15]. Meanwhile, according to the number of the backups, three types of primary-backup replication approaches exist; single backup for each primary, fixed ε backups for each primary, and quantitative backups for each primary.

The single backup for each primary approach is a simple method. Main representative methods include efficient fault-tolerant reliability cost driven (eFRCD) [33], efficient fault-tolerant reliability driven (eFRD) [34], and minimum completion time with less replication cost (MCT-LRC) [35] et al. Regarding their limitations, first, these approaches assume that no more than one failure happens at one moment; they are too ideal to tolerate potential multiple failures. Second, although passive replication also supports multiple backups for each primary [32], it is unsuitable for service-oriented applications; the reason is that once a processor failure is detected, the scheduler should reschedule the task located on the failed processor, and reassign it to a new processor, such that the QoS for the application is uncertain.

The fixed ε backups for each primary approach is an active replication approach, and is suitable for service-oriented systems because it can directly shield the failed tasks in performing, and the failure recovery time is almost close to zero [19], [20], [21]. In [19], the authors presented bicriteria scheduling heuristic (BSH) to minimize the schedule length of the application while taking the failure rate as a constraint; BSH can generate a Pareto curve of non-dominated solutions, among which the user can choose the compromise that fits his requirements best. However, the time complexity of BSH is as high as $O(n \times 2^u)$, where n is the number of replicas and u is the number of processors. In [20], Benoit et al. presented the fault-tolerant scheduling algorithm (FTSA) for a parallel application on heterogeneous systems to minimize the schedule length given a fixed number of failures supported in the system based on the active replication scheme. In [21], Benoit et al. further designed a new scheduling algorithm to minimize schedule length under both throughput and reliability constraints for a parallel application on heterogeneous systems based on the active replication scheme. The main problem in [20], [21] is that they need ε backups for each task with high redundancy to satisfy application's reliability requirement. Although application's reliability requirement can be satisfied by using active replication scheme, high redundancy causes high resource cost to resource providers.

Considering that fixed ε backups for each primary approach has high redundancy, recent studies begun to explore quantitative backups for each task approach to satisfy application's reliability requirement [14], [15]. Quantitative backups means different primaries have different numbers of backups, and the quantitative approach has lower resource cost than the fixed ε backups for each task based on active replication [14]. In [14] and [15], the authors proposed fault-tolerant scheduling algorithms MaxRe and RR; both MaxRe and RR incorporate reliability analysis into the active replication and exploit a dynamic number of backups for different tasks by considering each task's sub-reliability requirement. As discussed in Section 1.2, both MaxRe and RR have limitations in calculating the sub-reliability requirements of tasks. In [15], the authors also presented the DRR algorithm that extends RR by further considering the deadline requirement of a parallel application; however, we are only interested in satisfying reliability requirement in this study.

3 Reliability Modeling and Problem Statement

Table 1 gives the important notations and their definitions as used in this study.

3.1 Application Model

Let $U = \{u_1, u_2, \ldots, u_{|U|}\}$ represent a set of heterogeneous processors, where $|U|$ is the size of set U. In this study, for any set X, $|X|$ is used to denote size. A development life cycle of a service-oriented system usually involves the analysis, design, implementation, and testing phases. In this study, we focus on the design phase. Therefore, we assume that the processor and application parameter values are known in the design phase, because these values have been already calculated in the analysis phase.

TABLE 1
Important Notations in this Study

Notation	Definition
$c_{i,j}$	Communication time between the tasks n_i and n_j
$w_{i,k}$	Execution time of the task n_i on the processor u_k
$\overline{w_i}$	Average execution time of the task n_i
$rank_{\mathrm{u}}(n_i)$	Upward rank value of the task n_i
$\|X\|$	Size of the set X
λ_k	Constant failure rate per time unit of the processor u_k
num_i	Number of replicas of the task n_i
$NR(G)$	Total number of the replicas of the application G
$lb(n_i)$	Lower bound on number of replicas of the task n_i
n_i^x	xth replica of the task n_i
$u_{pr(n_i^x)}$	Assigned processor of the replica n_i^x
$R(n_i, u_k)$	Reliability of the task n_i on the processor u_k
$R(n_i)$	Reliability of the task n_i
$R(G)$	Reliability of the application G
$R_{\mathrm{seq}}(G)$	Reliability requirement of the application G
$R_{\mathrm{seq}}(n_i)$	Sub-reliability requirement of the task n_i
$R_{\mathrm{up\_seq}}(n_i)$	Upper bound on reliability requirement of the task n_i

As mentioned earlier, a parallel application running on processors is represented by a DAG G=(N, W, M, C) with known values.

(1) N represents a set of nodes in G, and each node $n_i \in N$ is a task with different execution time values on different processors. In addition, task executions of a given application are assumed to be non-preemptive which is possible in many systems [8], [14]. $pred(n_i)$ is the set of immediate predecessor tasks of n_i, while $succ(n_i)$ is the set of immediate successor tasks of n_i. Tasks without predecessor tasks are denoted by n_{entry}; and tasks with no successor tasks are denoted by n_{exit}. If an application has multiple entry or multiple exit tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph. W is an $|N| \times |U|$ matrix in which $w_{i,k}$ denotes the execution time of n_i running on u_k.

(2) M is a set of communication edges, and each edge $m_{i,j} \in M$ represents a communication from n_i to n_j. Accordingly, $c_{i,j} \in C$ represents the communication time of $m_{i,j}$ if n_i and n_j are assigned to different processors because two tasks with immediate precedence constraints need to exchange messages. When both tasks n_i to n_j are allocated to the same processor, $c_{i,j}$ becomes zero because we assume that the intra-processor communication cost is negligible [14], [15].

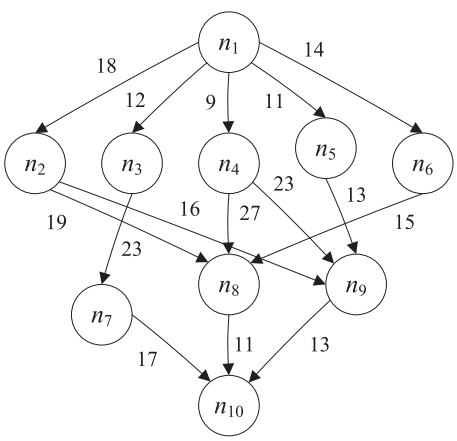


Fig. 2. Motivating example of a DAG-based parallel application with 10 tasks [9], [10], [11], [12].

TABLE 2
Execution Time Values of Tasks on Different Processors of the Motivating Parallel Application [9], [10], [11]

Task	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

Fig. 2 shows a motivating parallel application with tasks and messages [9], [10], [11], [12]. The example shows 10 tasks executed on 3 processors $\{u_1, u_2, u_3\}$. The weight 18 of the edge between n_1 and n_2 represents communication time, denoted by $c_{1,2}$ if n_1 and n_2 are not assigned to the same processor.

Table 2 is the execution time matrix $|N| \times |U|$ of tasks on different processors of the motivating parallel application. For example, the weight 14 of n_1 and u_1 in Table 2 represents execution time of n_1 on u_1, denoted by $w_{1,1}$=14. We can see that the same task has different execution time values on different processors due to the heterogeneity of the processors.

The motivating example will be used to explain the MaxRe, RR, and the proposed LBR, ERRM, and HRRM algorithms in the paper.

3.2 Reliability Model

There are two major types of failures: transient failure (also called random hardware failure) and permanent failure. Once a permanent failure occurs, the processor cannot be restored unless by replacement. The transient failure appears for a short time and disappear without damage to processors. Therefore, this paper mainly takes the transient failures into account for our research. In general, the occurrence of transient failure for a task in a DAG-based application follows the Poisson distribution [14], [15], [19], [31], [36]. The reliability of an event in unit time t is denoted by

$$R(t) = e^{-\lambda t},$$

where λ is the *constant failure rate per time unit* for a processor. We use λ_k to represent the constant failure rate per time unit of the processor u_k. The reliability of n_i executed on u_k in its execution time is denoted by

$$R(n_i, u_k) = e^{-\lambda_k w_{i,k}}, \quad (1)$$

and the failure probability for n_i without using the active replication is

$$1 - R(n_i, u_k) = 1 - e^{-\lambda_k w_{i,k}}. \quad (2)$$

However, each task has a number of replicas with the active replication. We define num_i ($num_i \leqslant |U|$) as the number of replicas of n_i. Hence, the replica set of n_i is $\{n_i^1, n_i^2, \ldots, n_i^{num_i}\}$ where n_i^1 is the primary and others are backups. As long as one replica of n_i is successfully completed, then we can recognize that there is no occurrence of failure for n_i, and the reliability of n_i is updated to

$$R(n_i) = 1 - \prod_{x=1}^{num_i} \left(1 - R\left(n_i^x, u_{pr(n_i^x)}\right)\right), \quad (3)$$

where $u_{pr(n_i^x)}$ represents the assigned processor of n_i^x. The difference between $R(n_i, u_k)$ and $R(n_i)$ is below: $R(n_i, u_k)$ is the value before task replication, whereas $R(n_i)$ is the value after task replication.

The reliability of the parallel application with precedence-constrained tasks should be [14]

$$R(G) = \prod_{n_i \in N} R(n_i). \quad (4)$$

In [15], both communication and computation failures are considered; however, some communication networks themselves provide fault-tolerance. For instance, routing information protocol (RIP) and open shortest path first (OSPF) are designed to reroute packets to ensure that they reach their destination [37]. Therefore, similar to some studies [14], [35], [38], this study only considers processor failure and excludes communication failure (i.e., the communication is assumed to be reliable in this study). In addition, we mainly focus on the redundancy minimization of tasks, which is not directly related to communication.

3.3 Problem Statement

As discussed in Section 1, any application cannot be 100 percent reliable, but if the system can satisfy application's reliability requirement, then the application is considered reliable. The problem addressed in this study can be formally described as follows. Assume that we are given a parallel application G and a heterogeneous processor set U. The problem is to assign replicas and corresponding processors for each task, while minimizing the number of replicas and ensuring that the obtained reliability of the application $R(G)$ satisfies the application's reliability requirement $R_{seq}(G)$. The formal description is to find the replicas and processor assignments of all tasks to minimize

$$NR(G) = \sum_{n_i \in N} num_i, \quad (5)$$

subject to

$$R(G) = \prod_{n_i \in N} R(n_i) \geqslant R_{\text{req}}(G),$$

for $\forall i : 1 \leqslant i \leqslant |N|$.

TABLE 3
Upward Rank Values for Tasks of the Motivating Parallel Application

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$rank_{\text{u}}(n_i)$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7

4 State-of-the-Art Approaches

4.1 Task Prioritizing

A fault-tolerant scheduling algorithm generally consists of three steps: 1) task prioritizing, 2) processor selection, and 3) task execution. Therefore, we should first compute the task priority before processor selection. Similar to state-of-the-art studies [14], [15], this study uses the famous upward rank value ($rank_{\text{u}}$) of a task (Eq. (6)) as the task priority standard. In this case, the tasks are ordered by descending order of $rank_{\text{u}}$, which are obtained by Eq. (6) [9], as follows:

$$rank_{\text{u}}(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{c_{i,j} + rank_{\text{u}}(n_j)\}, \quad (6)$$

in which $\overline{w_i}$ represents the average execution time of task n_i and is calculated as follows:

$$\overline{w_i} = \left(\sum_{k=1}^{|U|} w_{i,k}\right) \Big/ |U|.$$

Table 3 shows the upward rank values of all the tasks in Fig. 2. Note that only if all the predecessors of n_i have been assigned, will n_i prepare to be assigned. Assume that two tasks n_i and n_j satisfy $rank_{\text{u}}(n_i) > rank_{\text{u}}(n_j)$; if no precedence constraint exists between n_i and n_j, n_i does not necessarily take precedence for n_j to be assigned. Therefore, the task assignment order in G is $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

4.2 Existing MaxRe Algorithm

As the application reliability is the product of all the task reliability values, such problem is usually solved by transferring application's reliability requirement to the sub-reliability requirements of tasks [14], [15], [39]. In the MaxRe algorithm [14], the sub-reliability requirement for each task is calculated by

$$R_{\text{req}}(n_i) = \sqrt[|N|]{R_{\text{req}}(G)}. \quad (7)$$

If the sub-reliability requirement of each task can be satisfied by active replication below

$$R(n_i) \geqslant R_{\text{req}}(n_i),$$

then obviously the application's reliability requirement can be satisfied. Therefore, the main idea of the MaxRe algorithm is to iteratively select the replica n_i^x and processor $u_{pr(n_i^x)}$ with the maximum $R(n_i^x, u_{pr(n_i^x)})$ until the actual reliability value is larger than or equal to the sub-reliability requirement of the task, namely,

$$R(n_i) \geqslant R_{\text{req}}(n_i).$$

Moreover, this policy was also employed by the authors in [39].

TABLE 4
Task Assignment of the Motivating Parallel Application Using the MaxRe Algorithm

n_i	$R_{\text{req}}(n_i)$	$R(n_i,u_1)$	$R(n_i,u_2)$	$R(n_i,u_3)$	num_i	$R(n_i)$
n_1	0.99383156	**0.98609754**	0.97628571	**0.98393051**	2	0.99977659
n_3	0.99383156	**0.98906028**	**0.98068890**	0.96637821	2	0.99978874
n_4	0.99383156	**0.98708414**	**0.98807171**	0.96986344	2	0.99984594
n_2	0.99383156	**0.98708414**	**0.97190229**	0.96811926	2	0.99963709
n_5	0.99383156	**0.98807171**	0.98068890	**0.98216103**	2	0.99978721
n_6	0.99383156	**0.98708414**	0.97628571	**0.98393051**	2	0.99979245
n_9	0.99383156	**0.98216103**	**0.98216103**	0.96464029	2	0.99968177
n_7	0.99383156	**0.99302444**	0.97775124	**0.98039473**	2	0.99986324
n_8	0.99383156	**0.99501248**	0.98363538	0.97511487	1	0.99501248
n_{10}	0.99383156	**0.97921896**	**0.98955493**	0.97161077	2	0.99978294
$NR(G)=19$, $R(G)=0.99298048$						

TABLE 5
Task Assignment of the Motivating Parallel Application Using the RR Algorithm

n_i	$R_{\text{req}}(n_i)$	$R(n_i,u_1)$	$R(n_i,u_2)$	$R(n_i,u_3)$	num_i	$R(n_i)$
n_1	0.99383156	**0.98609754**	0.97628571	**0.98393051**	2	0.99977659
n_3	0.99317319	**0.98906028**	**0.98068890**	0.96637821	2	0.99978874
n_4	0.99234932	**0.98708414**	**0.98807171**	0.96986344	2	0.99984594
n_2	0.99128298	**0.98708414**	**0.97190229**	0.96811926	2	0.99963709
n_5	0.98989744	**0.98807171**	0.98068890	**0.98216103**	2	0.99978721
n_6	0.98793125	**0.98708414**	0.97628571	**0.98393051**	2	0.99979245
n_9	0.98498801	**0.98216103**	**0.98216103**	0.96464029	2	0.99968177
n_7	0.98013824	**0.99302444**	0.97775124	0.98039473	1	0.99302444
n_8	0.97511487	**0.99501248**	0.98363538	0.97511487	1	0.99501248
n_{10}	0.97161077	0.97921896	**0.98955493**	0.97161077	1	0.98955493
$NR(G)=17$, $R(G)=0.97609982$						

Example 1. Assume that the constant failure rates for three processors are $\lambda_1 = 0.0010$, $\lambda_2 = 0.0015$, and $\lambda_3 = 0.0018$, respectively. Moreover, assume that the reliability requirement of the parallel application in Fig. 2 is $R_{\text{seq}}(G) = 0.94$. Note that the above values are not the representatives of a real deployment, but are used to explain the example clearly.

Table 4 shows the task assignment for each task of the motivating parallel application using the MaxRe algorithm. Each row shows the selected processors (denoted with bold text) and corresponding reliability values. For example, the sub-reliability requirement of n_1 is $R_{\text{req}}(n_1) = \sqrt[10]{0.94} = 0.99383156$; to satisfy the sub-reliability requirement, MaxRe selects the processors u_1 and u_3 with the maximum and second maximum reliability values, respectively (i.e., $num_1 = 2$). Then, the actual reliability value of n_1 is 0.99977659, which is calculated by Eq. (3). The remaining tasks use the same pattern with n_1. Finally, the number of replicas are 19 and the actual reliability value of the application G is 0.99298048, which are calculated by Eqs. (5) and (4), respectively.

4.3 Existing RR Algorithm

Obviously, the main limitation of the MaxRe algorithm is that the sub-reliability requirements of all tasks are equal and high, such that it needs more replicas with extra redundancy to satisfy the sub-reliability requirement of each task. To solve such problem, the authors presented the RR algorithm to lower down the sub-reliability requirement of tasks while still satisfying the application's reliability requirement [15] as follows.

First, the sub-reliability requirement for the entry task is still calculated by

$$R_{\text{req}}(n_1) = \sqrt[|N|]{R_{\text{req}}(G)}.$$

Second, for the rest of tasks (i.e., non-entry tasks), unlike prior MaxRe algorithm [14], sub-reliability requirements in the RR algorithm are calculated continuously based on the actual reliability achieved by previous allocations

$$R_{\text{req}}(n_{seq(j)}) = \sqrt[|N|-j+1]{\frac{R_{\text{req}}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)})}}, \tag{8}$$

where $n_{seq(j)}$ represents the jth assigned task. Clearly, such single improvement can reduce the sub-reliability requirements of non-entry tasks.

Example 2. The same parameter values ($\lambda_1 = 0.0010$, $\lambda_2 = 0.0015$, $\lambda_3 = 0.0018$, and $R_{\text{seq}}(G) = 0.94$) with Example 1 are used. Table 5 shows the task assignment for each task of the motivating parallel application using the RR algorithm. Each row shows the selected processors (denoted with bold text) and corresponding reliability values. The sub-reliability requirement and task assignment of n_1 using the RR algorithm is similar to the MaxRe algorithm. However, the remaining tasks are different. For example, as the actual reliability value for n_1 is 0.99977659, then the sub-reliability requirement for n_3 should be $\sqrt[9]{\frac{0.94}{0.99977659}} = 0.99317319$. When assigning n_7 and n_8, the sub-reliability requirements are reduced to 0.98013824 and 0.97161077, respectively. That is, only one replica for each of n_7 and n_8 will be able to satisfy individual sub-reliability requirements. Finally, the number of replicas and the actual reliability value of the application G are 17 and 0.97609982 (calculated by Eqs. (5) and (4)), respectively, which still satisfy application's reliability requirement, but their values are less than those obtained with the MaxRe algorithm.

5 Enough Replication for Redundancy Minimization

Although the RR algorithm can reduce the sub-reliability requirements of tasks, the reduction ranges of tasks near the entry task are much lower than those of the tasks near the exit task. That is, the actual sub-reliability requirements show unfairness among tasks, such that the RR algorithm still requires unnecessary redundancy to satisfy application's reliability requirement. To further reduce redundancy, we first present good enough replication approach in this section, and then propose a heuristic replication approach in the next section.

5.1 Lower Bound on Redundancy

Considering that application reliability is the product of all task reliability values, the reliability value of each task should be higher than or equal to $R_{\text{req}}(G)$; otherwise, if one task has

$R(n_i) < R_{\text{req}}(G)$, then no matter how many replicas for any other tasks, $R_{\text{req}}(G)$ cannot be satisfied. Therefore, the lower bound on reliability requirement of the task n_i is

$$R_{\text{lb\_req}}(n_i) = R_{\text{req}}(G). \tag{9}$$

In this way, there should be a lower bound on the number of replicas for each task that satisfies

$$R(n_i) \geqslant R_{\text{lb\_req}}(n_i).$$

In other words, we can determine the lower bound on the number of replicas $lb(n_i)$ for task n_i to satisfy

$$1 - \prod_{x=1}^{lb(n_i)} (1 - R(n_i^x, u_{pr(n_i^x)})) \geq R_{\text{lb\_req}}(n_i), \tag{10}$$

according to Eq. (3).

We use the following steps to select the replica and the corresponding processor with the minimum number of replicas.

(1) Calculate the $R(n_i, u_k)$ of each task on all available processors (if a replica of n_i has been assigned to the processor, then this processor is unavailable for n_i; otherwise, it is available for n_i).
(2) To minimize the number of replicas, select the replica n_i^x of the task n_i and the corresponding processor $u_{pr(n_i^x)}$ with the maximum $R(n_i^x, u_{pr(n_i^x)})$.
(3) Repeat Steps (1) and (2) until Eq. (10) is satisfied.

5.2 The LBR Algorithm

On the basis of the above steps, we propose the lower bound on redundancy (LBR) algorithm (Algorithm 1) to generate the lower bound on the number of the replicas of each task.

Algorithm 1. The LBR Algorithm

Input: $G = (N, W, M, C)$, U, $R_{\text{req}}(G)$
Output: $R(G)$, $NR(G)$ and its related values
1: **for** $(i = 1; i <= |N|; i++)$ **do**
2: $R_{\text{lb\_req}}(n_i) \leftarrow R(G)$;
3: $num_i = 0$;
4: $R(n_i) = 0$; // initial value is 0
5: **while** $(R(n_i) < R_{\text{lb\_req}}(n_i))$ **do**
6: Calculate $R(n_i, u_k)$ for the task n_i on all available processors using Eq. (1);
7: Select replica n_i^x and the processor $u_{pr(n_i^x)}$ with the maximum reliability value $R(n_i^x, u_{pr(n_i^x)})$;
8: num_i++;
9: Calculate $R(n_i)$ using Eq. (3);
10: **end while**
11: Calculate $NR(G)$ using Eq. (5);
12: Calculate $R(G)$ using Eq. (4);
13: **end for**

The core idea of the LBR algorithm is that each task iteratively selects the replica and available processor with the maximum reliability value $R(n_i^x, u_{pr(n_i^x)})$ for each task until the task's lower bound on reliability requirement is satisfied. The details are explained as follows:

TABLE 6
Task Assignment of the Motivating Parallel Application Using the LBR Algorithm

n_i	$R_{\text{req}}(n_i)$	$R(n_i, u_1)$	$R(n_i, u_2)$	$R(n_i, u_3)$	num_i	$R(n_i)$
n_1	0.94	**0.98609754**	0.97628571	0.98393051	1	0.98609754
n_3	0.94	**0.98906028**	0.98068890	0.96637821	1	0.98906028
n_4	0.94	0.98708414	**0.98807171**	0.96986344	1	0.98807171
n_2	0.94	**0.98708414**	0.97190229	0.96811926	1	0.98708414
n_5	0.94	**0.98807171**	0.98068890	0.98216103	1	0.98807171
n_6	0.94	**0.98708414**	0.97628571	0.98393051	1	0.98708414
n_9	0.94	**0.98216103**	0.98216103	0.96464029	1	0.98216103
n_7	0.94	**0.99302444**	0.97775124	0.98039473	1	0.99302444
n_8	0.94	**0.99501248**	0.98363538	0.97511487	1	0.99501248
n_{10}	0.94	0.97921896	**0.98955493**	0.97161077	1	0.98955493

$NR(G) = 10$, $R(G) = 0.89092057$

(1) In Line 2, LBR has obtained the lower bound on reliability requirement of the current task before it prepares to be assigned.
(2) In Lines 5-10, LBR iteratively selects the replica and available processor for each task with the maximum reliability value until the task's lower bound on reliability requirement is satisfied. Specifically, the following details are made: 1) Line 5 compares the actual reliability value and lower bound on reliability requirement of the current task; 2) Lines 6-7 calculate and select the replica and available processor with the maximum reliability value for the current task; and 3) Line 9 calculates the actual reliability value of the current task.
(3) In Lines 11-12, LBR calculates the final number of replicas and the actual reliability value of the application, respectively.

5.3 Time Complexity of the LBR Algorithm

The time complexity of the LBR algorithm is analyzed as follows:

(1) Calculating the reliability of the application must traverse all tasks, which can be done within O($|N|$) time (Lines 1-13).
(2) The total number of replicas for each task must be lower or equal to the number of processors, which can be done within O($|U|$) time (Lines 5-10).
(3) Selecting the replica and available processor with the maximum reliability value for the current task must traverse all processors, which can be done in O($\log|U|$) time (Line 7).

Thus, the time complexity of the LBR algorithm is O($|N| \times |U| \times \log|U|$).

5.4 Example of the LBR Algorithm

Example 3. The same parameter values ($\lambda_1 = 0.0010$, $\lambda_2 = 0.0015$, $\lambda_3 = 0.0018$, and $R_{\text{seq}}(G) = 0.94$) with aforementioned examples are used. Table 6 lists the replicas, selected processor, and reliability value of each task (denoted with bold text). We find that the reliability value of each task is higher than the application's reliability requirement of 0.94. However, the current obtained reliability value of the

parallel application is only $R(G) = 0.89092057$ (calculated by Eq. (4)), which is much lower than 0.94 (application's reliability requirement). Hence, application's reliability requirement is not satisfied by merely using the LBR algorithm.

5.5 Enough Replication

Considering that all the tasks merely satisfy $R(n_i) \geq R_{\text{lb\_req}}(n_i)$ by using the LBR algorithm (Algorithm 1), we should add more new replicas for tasks to satisfy application's reliability requirement. However, choosing the remaining replicas is a complex work, because different replicas of different tasks may cause different reliability values on different processors.

Given that the current number of replicas for n_i is $h = num_i$ and the application reliability is $R(G)$, if a new replica n_i^{h+1} is assigned to the processor $u_k = u_{pr(n_i^{h+1})}$ for n_i, then the number of replicas is changed to $h+1$ and the new task reliability is changed to

$$R_{\text{new}}(n_i) = 1 - \prod_{x=1}^{h+1} (1 - R(n_i^x, u_{pr(n_i^x)})). \tag{11}$$

Then, the application reliability is enhanced because of the reliability enhancement of n_i and is changed to

$$R^i(G) = R_{\text{new}}(n_i) \times \prod_{n_j \in N, i \neq j} R(n_j). \tag{12}$$

To minimize the number of replicas for each task, we use the following steps to obtain enough minimum redundancy of the application.

(1) Each available task (if the replicas of a task have been assigned to all the processors, then this task is unavailable; otherwise, a task is available) is assumed to be replicated once on an available processor with the maximum $R(n_i, u_k)$ (Eq. (1)), and the new task sub-reliability is changed to $R_{\text{new}}(n_i)$ (Eq. (11)).

(2) Calculate the application reliability $R^i(G)$ because of the reliability enhancement of each task (Eq. (12)).

(3) Select the replica n_i^x and corresponding processor $u_{pr(n_i^x)}$ that generate the maximum $R^i(G)$ from the generated replicas in Step 2), namely,

$$R^i(G) = \max\Big\{R^1(G), R^2(G), \ldots, R^{|N|}(G)\Big\}. \tag{13}$$

(4) Repeat Steps (1), (2), and (3) until application's reliability requirement (Eq. (4)) is satisfied.

5.6 The ERRM Algorithm

In this section, we propose the ERRM algorithm to minimize redundancy to satisfy application's reliability requirement, and describe the steps in Algorithm 2.

The core idea of the ERRM algorithm is that all the tasks are first assumed to be replicated once on an available processor with the maximum reliability values; then ERRM selects the replica n_s^x and corresponding processor $u_{pr(n_s^x)}$ that generate the maximum application reliability value $R^s(G)$ until application's reliability requirement is satisfied in the iterative replication process. The details are explained as follows:

(1) In Line 1, ERRM calls the LBR algorithm (Algorithm 1) to obtain the initial reliability $R(G)$ and related values.

(2) In Lines 2-11, ERRM iteratively selects the replica and available processor that generate the maximum application reliability value until application's reliability requirement is satisfied. Specifically, the following details are made: 1) Line 2 compares the actual reliability value and the reliability requirement of the application; 2) Lines 3-7 pre-replicate all tasks once on an available processor with the maximum reliability values; 3) Line 8 selects the replica and corresponding processor that generate the maximum application reliability value; and 4) Line 10 updates the application's reliability value.

(3) In Line 13, ERRM calculates the final number of replicas of the application.

Algorithm 2. The ERRM Algorithm

Input: $G = (N, W, M, C)$, U, $R_{\text{req}}(G)$
Output: $R(G)$, $NR(G)$ and its related values
1: Call the LBR algorithm (Algorithm 1) to obtain the initial reliability $R(G)$ and related values;
2: **while** $(R(G) < R_{\text{req}}(G))$ **do**
3: **for** $(i = 1; i <= |N|; i++)$ **do**
4: Pre-replicated the replica of n_i on an available processor with the maximum reliability value $R(n_i, u_k)$;
5: Update the task's sub-reliability value to $R_{\text{new}}(n_i)$ (Eq. (11));
6: Calculate the application reliability $R^i(G)$ after the reliability enhancement of n_i (Eq. (12));
7: **end for**
8: Select the replica n_s^x and corresponding processor $u_{pr(n_s^x)}$ that generate the maximum application reliability value $R^s(G)$ (Eq. (13));
9: num_i++;
10: $R(n_i) \leftarrow R_{\text{new}}(n_i)$;
11: $R(G) \leftarrow R^i(G)$;
12: **end while**
13: Calculate $NR(G)$ using Eq. (5);

5.7 Time Complexity of the ERRM Algorithm

The time complexity of the ERRM algorithm is analyzed as follows:

(1) The maximum number of iterative replication process is $|N| \times |U|$, which can be done within $O(|N| \times |U|)$ time (Lines 2-12).

(2) Each task must be assumed to be replicated once on an available processor, which can be done in $O(|N|)$ time (Lines 3-7).

(3) Selecting the replica and available processor with the maximum reliability value must traverse all processors, which can be done in $O(\log|U|)$ time (Line 4).

(4) Updating the task's new sub-reliability value can be done in $O(|U|)$ time (Line 5).

(5) Calculating the application's new reliability value can be done in $O(|N|)$ time (Line 6).

(6) Obtaining the maximum application reliability value can be done in $O(|N|)$ time (Line 8).

Considering that (3), (4), and (5) are not nested in the algorithm, the time complexity of the ERRM algorithm is $O(|N|^2 \times |U|^2 + |N|^3 \times |U|)$.

TABLE 7
Selected Processor and Reliability Pairs (Denoted with Underline Text) of Each Task in Each Step of the Motivating Parallel Application Using the ERRM Algorithm

Step	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
(1)	$(u_3, 0.9033)$	$(u_2, 0.9023)$	$(u_2, 0.9006)$	$(u_1, 0.9015)$	$(u_3, 0.9015)$	$(u_3, 0.9024)$	$(u_3, 0.89714)$	$(u_2, 0.8953)$	$(u_2, 0.9068)$	$(p_1, 0.9001)$
(2)	$(u_3, 0.9194)$	$(u_2, 0.9183)$	$(u_2, 0.9166)$	$(u_1, 0.9176)$	$(u_3, 0.9176)$	$(u_3, 0.9185)$	$(u_2, 0.9131)$	$(u_2, 0.9113)$	$(u_3, 0.9071)$	$(u_1, 0.9162)$
(3)	$(u_2, 0.9196)$	$(u_2, 0.9311)$	$(u_2, 0.9294)$	$(u_1, 0.9303)$	$(u_3, 0.9303)$	$(u_3, 0.9312)$	$(u_3, 0.9257)$	$(u_2, 0.9239)$	$(u_3, 0.9197)$	$(u_1, 0.9289)$
(4)	$(u_2, 0.9314)$	$(u_2, 0.9431)$	$(u_2, 0.9413)$	$(u_1, 0.9423)$	$(u_3, 0.9423)$	$(u_2, 0.9314)$	$(u_3, 0.9376)$	$(u_2, 0.9358)$	$(u_3, 0.9315)$	$(u_1, 0.9409)$

Considering that the time complexity of the LBR algorithm (i.e., $O(|N| \times |U| \times \log|U|)$) is less than that of the ERRM algorithm, using the LBR algorithm in advance can improve the efficiency of the replication compared with only using the ERRM algorithm. The reason is that the LBR algorithm can obtain an initial reliability value greater than zero, such that the number of iterative process of the ERRM algorithm can be reduced. Considering the motivating example, the reliability value obtained is 0.89092057, shown in Table 6, then the initial reliability value is not 0, but 0.89092057. Compared to starting from 0, 0.89092057 is close to the actual reliability requirement of 0.93.

5.8 Example of the ERRM Algorithm

Example 4. The same parameter values ($\lambda_1 = 0.0010$, $\lambda_2 = 0.0015$, $\lambda_3 = 0.0018$, and $R_{\text{seq}}(G) = 0.94$) with aforementioned examples are used. Table 7 lists the selected processor and reliability pairs of each task in each step by using Algorithm 2, where the underlined values indicate those that have the maximum $R_{\text{new}}(n_i)$ (Eq. (11)) and $R^{\text{s}}(G)$ (Eq. (13)), and the replica is selected to enhance the reliability of the application in each step. For example, in Step (1), n_9 and u_2 are selected, because they can generate the maximum value of 0.9068. In Step (4), the reliability value is larger than or equal to application's reliability requirement 0.94. Hence, application's reliability requirement is satisfied, and the replication process successfully ends.

Table 8 lists the final replicas, selected processor, and reliability value for each task of the parallel application in Fig. 2. We find that the final reliability value of each task is larger than or equal to 0.94. Moreover, the current reliability value is $R(G) = 0.94307237$ (calculated by Eq. (4)), which is larger than 0.94. Hence, application's reliability requirement is satisfied, and the application proves reliable in this situation. Meanwhile, the final resource consumption is $NR(G) = 14$ (Calculated by Eq. (5)).

TABLE 8
Task Assignment of the Application in Fig. 2 Using the ERRM Algorithm

n_i	$R(n_i, u_1)$	$R(n_i, u_2)$	$R(n_i, u_3)$	num_i	$R(n_i)$
n_1	**0.98609754**	0.97628571	0.98393051	2	0.99977659
n_3	**0.98906028**	0.98068890	0.96637821	1	0.98906028
n_4	0.98708414	**0.98807171**	0.96986344	1	0.98807171
n_2	**0.98708414**	0.97190229	0.99963709	2	0.98708414
n_5	**0.98807171**	0.98068890	0.98216103	1	0.98807171
n_6	**0.98708414**	0.97628571	0.98393051	2	0.99979245
n_9	**0.98216103**	0.98216103	0.96464029	2	0.99968177
n_7	**0.99302444**	0.97775124	0.98039473	1	0.99302444
n_8	**0.99501248**	0.98363538	0.97511487	1	0.99501248
n_{10}	0.97921896	**0.98955493**	0.97161077	1	0.98955493
$NR(G) = 14$, $R(G) = 0.94307237$					

6 Heuristic Replication for Redundancy Minimization

Although the ERRM algorithm can implement enough redundancy minimization, it has high time complexity and thereby it is time-consuming for a large-scale parallel application. To reduce the redundancy of a large-scale parallel application within an acceptable computation time, this section presents a heuristic algorithm.

6.1 Upper Bound on Reliability Requirement

Although the RR algorithm can achieve more redundancy reduction than the MaxRe algorithm by recalculating the sub-reliability requirement, the redundancy reduction ranges of the tasks near the entry task is much lower than those of the tasks near the exit task (see Table 5). The main reason for the discrepancy is that unfair sub-reliability requirements among tasks are generated. In fact, the tasks that are after $n_{seq(x)}$'s allocations (i.e., unassigned tasks) can also be presupposed as assigned tasks with known reliability values.

We find that all the sub-reliability requirements of tasks using the RR algorithm do not exceed 0.99383156 (see Table 5), which is the sub-reliability requirement of each task using the MaxRe algorithm (see Table 4). Thus, we let $\sqrt[|N|]{R_{\text{req}}(G)}$ be the upper bound on task's reliability requirement, namely,

$$R_{\text{up\_req}}(n_i) = \sqrt[|N|]{R_{\text{req}}(G)}. \tag{14}$$

Then, we have the following heuristic strategy: assume that the task to be assigned is $n_{seq(j)}$ ($n_{seq(j)}$ represents the jth assigned task as mentioned earlier), then $\{n_{seq(1)}, n_{seq(2)}, \ldots, n_{seq(j-1)}\}$ represents the task set with assigned tasks, and $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$ represents the task set with unassigned tasks. To ensure that the reliability of the application is satisfied at each task assignment, we presuppose that each task in $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$ is assigned to the processor with reliability value on upper bound (Eq. (14)). Hence, when assigning $n_{seq(j)}$, application's reliability requirement is

$$R_{\text{req}}(G) = \prod_{x=1}^{j-1} R(n_{seq(x)}) \times R_{\text{req}}(n_{seq(j)}) \times \prod_{y=j+1}^{|N|} R_{\text{up\_req}}(n_{seq(y)}).$$

Then, the sub-reliability requirement for the task $n_{seq(j)}$ should be

$$R_{\text{req}}(n_{seq(j)}) = \frac{R_{\text{req}}(G)}{\prod_{x=1}^{j-1} R(n_{seq(x)}) \times \prod_{y=j+1}^{|N|} R_{\text{up\_req}}(n_{seq(y)})}. \tag{15}$$

6.2 The HRRM Algorithm

On the basis of the aforementioned new sub-reliability requirement calculation for each task (Eq. (15)), we present the heuristic algorithm HRRM described in Algorithm 3 to minimize redundancy and satisfy application's reliability requirement.

Algorithm 3. The HRRM Algorithm

Input: $G = (N, W, M, C)$, U, $R_{\text{req}}(G)$
Output: $R(G)$, $NR(G)$ and its related values
1: Order tasks according to a descending order of $rank_{\text{u}}(n_i, u_k)$ using Eq. (6);
2: **for** $(j = 1; j <= |N|; j++)$ **do**
3: Calculate $R(n_{seq(j)})$ using Eq. (3);
4: $num_{seq(j)} = 0$;
5: $R(n_{seq(j)}) = 0$; // initial value is 0
6: Calculate $R_{\text{req}}(n_{seq(j)})$ using Eq. (15);
7: **while** $(R(n_{seq(j)}) < R_{\text{req}}(n_{seq(j)}))$ **do**
8: Calculate $R(n_{seq(j)}, u_k)$ for the task $n_{seq(j)}$ on all each available processor using Eq. (1);
9: Select replica $n^x_{seq(j)}$ and the processor $u_{pr(n^x_{seq(j)})}$ with the maximum $R(n^x_{seq(j)}, u_{pr(n^x_{seq(j)})})$;
10: $num_{seq(j)}$++;
11: Calculate $R(n_{seq(j)})$ using Eq. (3);
12: **end while**
13: **end for**
14: Calculate $NR(G)$ using Eq. (5);
15: Calculate $R(G)$ using Eq. (4);

The core idea of HRRM is that the reliability requirement of the application is transferred to the sub-reliability requirement of each task. Each task just iteratively selects the replica and available processor with the maximum reliability value until its sub-reliability requirement is satisfied. The details are explained as follows:

(1) In Line 6, HRRM has obtained the reliability requirement of the current task before it prepares to be assigned.
(2) In Lines 7-12, HRRM iteratively selects the replica and available processor with the maximum reliability value for the current task until its sub-reliability requirement is satisfied. Specifically, the following details are made: 1) Line 7 compares the actual reliability value and sub-reliability requirement of the current task; 2) Lines 8-9 calculate and select the replica and available processor with the maximum reliability value for the current task; and 3) Line 11 calculates the actual reliability value of the current task.
(3) In Lines 14-15, HRRM calculates the final number of replicas and the actual reliability value of the application, respectively.

Compared with MaxRe and RR algorithms, the main improvement of the presented HRRM is that it recalculates the sub-reliability requirement of each task based not only on its previous assignments ($\{n_{seq(1)}, n_{seq(2)}, \ldots, n_{seq(j-1)}\}$), but also on succeeding pre-assignments $\{n_{seq(j+1)}, n_{seq(j+2)}, \ldots, n_{seq(|N|)}\}$, whereas MaxRe algorithm has a fixed and equal sub-reliability requirements for all tasks and RR algorithm is merely based on previous assignments.

TABLE 9
Task Assignment of the Motivating Parallel Application Using the HRRM Algorithm

n_i	$R_{\text{req}}(n_i)$	$R(n_i, u_1)$	$R(n_i, u_2)$	$R(n_i, u_3)$	num_i	$R(n_i)$
n_1	0.99383156	**0.98609754**	0.97628571	**0.98393051**	2	0.99977659
n_3	0.98792188	**0.98906028**	0.98068890	0.96637821	1	0.98906028
n_4	0.99268768	**0.98708414**	**0.98807171**	0.96986344	2	0.99984594
n_2	0.98671636	**0.98708414**	0.97190229	0.96811926	1	0.98708414
n_5	0.99346128	**0.98807171**	0.98068890	**0.98216103**	2	0.99978721
n_6	0.98754331	**0.98708414**	0.97628571	**0.98393051**	2	0.99979245
n_9	0.98165546	**0.98216103**	0.98216103	0.96464029	1	0.98216103
n_7	0.99331998	**0.99302444**	0.97775124	**0.98039473**	2	0.99986324
n_8	0.98732777	**0.99501248**	0.98363538	0.97511487	1	0.99501248
n_{10}	0.98615598	0.97921896	**0.98955493**	0.97161077	1	0.98955493
$NR(G) = 15$, $R(G) = 0.94323987$						

6.3 Time Complexity of the HRRM Algorithm

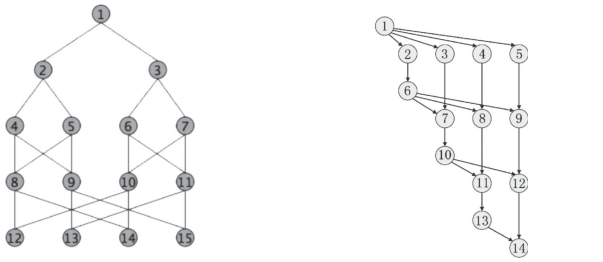
The time complexity of the HRRM algorithm is analyzed as follows:

(1) Calculating the reliability of the application must traverse all tasks, which can be done within $O(|N|)$ time (Lines 2-13).
(2) Calculating the sub-reliability requirement of the current task must traverse all tasks, which can be done within $O(|N|)$ time (Line 6).
(3) The number of replicas must be lower or equal to the number of processors, which can be done within $O(|U|)$ time (Lines 7-12).
(4) Calculating the reliability value of the current task must traverse all assigned processors, which can be done in $O(|U|)$ time (Line 11)

Considering that (2) and (3) are not nested in the algorithm, the time complexity of the HRRM algorithm is $O(|N|^2 + |N| \times |U|^2)$, which is similar to those of MaxRe and RR algorithms. Thus, HRRM implements efficient fault-tolerance without increasing time complexity.

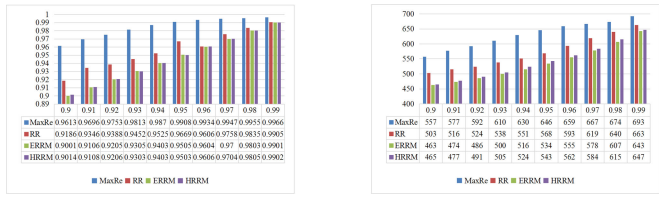
6.4 Example of the HRRM Algorithm

Example 5. The same parameter values ($\lambda_1 = 0.0010$, $\lambda_2 = 0.0015$, $\lambda_3 = 0.0018$, and $R_{\text{seq}}(G) = 0.94$) with aforementioned examples are used. Table 9 shows the task assignment for each task of the motivating parallel application using HRRM algorithm. Each row shows the selected processors (in red) and corresponding reliability values. The sub-reliability requirement and task assignment of n_1 using HRRM algorithm is similar to those using MaxRe and RR algorithms. However, the remaining tasks are different. For example, when assigning n_3, the actual reliability value for n_1 is 0.99977659, and succeeding pre-assignments with reliability requirements are $\sqrt[10]{0.94} = 0.99383156$, then the sub-reliability requirement for n_3 should be $\frac{0.94}{0.99977659 \times 0.99383156^8} = 0.98792188$. Compared with the RR algorithm, an obvious improvement for the HRRM algorithm is that it shows relative fair reliability requirements among tasks; furthermore, most sub-reliability requirements of tasks using HRRM are less than those using the RR algorithm. Finally, the number of replicas and the actual reliability value of the application



(a) Fast Fourier transform with ρ=4. (b) Gaussian elimination with ρ=5.

Fig. 3. Example of real parallel applications.



(a) Actual reliability values. (b) Total numbers of replicas.

Fig. 4. Results of the small-scale fast Fourier transform application on different reliability requirements (Experiment 1).

G are 15 and 0.94323987 (calculated by Eqs. (5) and (4)), respectively, which are lower than those with MaxRe and RR algorithms.

7 Experiments

7.1 Experimental Metrics and Parameter Values

Considering that this study aims to implement redundancy minimization with replication to satisfy application's reliability requirement, performance metrics selected for comparison should be the actual reliability value and total number of replicas of the application. Meanwhile, computation time should be included from a time complexity perspective. The computation time is measured from the start time to the end time of an algorithm to schedule an application.

Algorithms compared with the proposed ERRM and HRRM algorithms are the state-of-the-art MaxRe [14] and RR [15] algorithms. MaxRe and RR algorithms address the same problem of minimizing resource redundancy of a parallel application to satisfy application's reliability requirement on heterogeneous distributed systems.

Considering that this study focuses on the design phase, the processor and application parameters used in this phase are known. In other words, these values have been obtained in the analysis phase and are as follows [15]: 10,000 s $\leqslant w_{i,k} \leqslant$ 100,000 s, 10,000 s $\leqslant c_{i,j} \leqslant$ 100,000 s, and $0.000001 \leqslant \lambda_k \leqslant 0.000009$. The aforementioned values are generated with uniform distribution.

The parallel applications will be tested on a simulated heterogeneous system based on the above real processor and application parameter values to reflect a real deployment. A main advantage of simulation is that it can greatly reduce development cost during the design phase and effectively provide certain optimization guide to the implementation phase. The simulated multiprocessor system is configured 64 heterogeneous processors by creating 64 processor objects based known parameter values using Java on a standard desktop computer with 2.6 GHz Intel CPU and 4 GB memory.

Meanwhile, real parallel applications with precedence constrained tasks, such as fast Fourier transform and Gaussian elimination applications, are widely used in distributed systems [9], [15]. The Fourier transform and Gaussian elimination application are two typical parallel applications with high and low parallelism, respectively. To verify the effectiveness and validity of the proposed algorithms, we use the two types of real parallel applications to compare the results of all the algorithms.

A new parameter ρ is used as the size of the fast Fourier transform application. The total number of tasks is $|N| = (2 \times \rho - 1) + \rho \times \log_2 \rho$, where $\rho = 2^y$ for some integer y [9]. Fig. 3a shows an example of the fast Fourier transform application with ρ=4. Notably, ρ exit tasks exist in the fast Fourier transform application with the size of ρ. To adopt the application model of this study, we add a virtual exit task, and the last ρ tasks are set as the immediate predecessor tasks of the virtual exit task. A new parameter ρ is used as the matrix size of the Gaussian elimination application, and the total number of tasks is $|N| = \frac{\rho^2+\rho-2}{2}$ [9]. Fig. 3b shows an example of the Gaussian elimination parallel application with ρ=5.

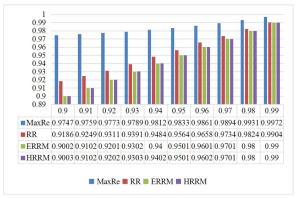
7.2 Fast Fourier Transform Application

Experiment 1. This experiment compares the actual reliability values and the total number of replicas of a small-scale fast Fourier transform application with $\rho = 32$ (i.e., $|N| = 223$) for varying reliability requirements. $R_{\text{seq}}(G)$ is changed from 0.9 to 0.99 with 0.01 increments. Note that computation time values of all the algorithms are within one second for the small-scale application and we no longer list such values in this experiment.

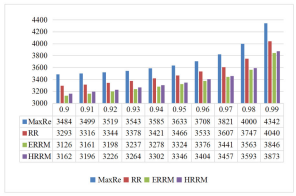
Note that the plotted values in Figs. 4a and 4b are obtained by executing one run of the algorithms for one application. Many applications with the same parameter values and scales are tested and show the same regular pattern and relatively stable results as Figs. 4a and 4b. In other words, experiments are repeatable and do not affect the consistency of the results. Therefore, the plotted values are the actual values rather than the average values during the runs.

Fig. 4a shows the actual reliability values of the small-scale fast Fourier transform application on different reliability requirements. We can see that all the algorithms can satisfy the given reliability requirements in all cases. Specifically, MaxRe generates the maximum reliability values followed by RR, HRRM, and ERRM. The overrunning reliability values (i.e., $R_{\text{seq}}(G)$-$R(G)$) reach 0.0613 and 0.0246 for MaxRe and RR, respectively. On the contrary, the overrunning reliability values are very small for HRRM (0.0001-0.0008) and ERRM (0.0001-0.0006) in all cases. Considering no additional fees will be paid for the overrunning reliability values, more resources are wasted for resource providers in using MaxRe and RR.

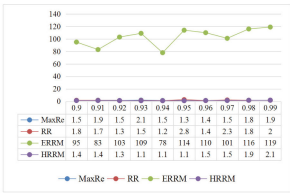
Fig. 4b shows the total number of replicas of the small-scale fast Fourier transform application on different reliability requirements. As expected, MaxRe generates the maximum numbers of replicas followed by RR, HRRM, and ERRM in all cases. The reason is that MaxRe has obtained the maximum actual reliability values followed by RR, HRRM, and ERRM in Fig. 4a, whereas optimizing reliability and



(a) Actual reliability values.



(b) Total numbers of replicas.



(c) Computation time (unit: second).

Fig. 5. Results of the large-scale fast Fourier transform application on different reliability requirements (Experiment 2).

redundancy is a bi-criteria optima problem as discussed in Section 1.2.

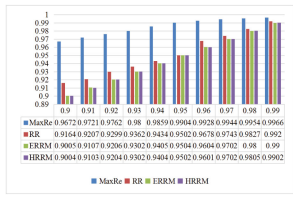
The same regular pattern for the actual reliability values is shown in Fig. 4a. As evident from Fig. 4b, the numbers of replicas using HRRM and ERRM are very similar and are much lower than those using MaxRe and RR, especially on relatively low reliability requirements. For example, when $R_{\text{seq}}(G) \leqslant 0.94$, both ERRM and HRRM outperform MaxRe and RR by about 18 and 7 percent, respectively.

Experiment 2. This experiment compares the actual reliability values, the total number of replicas, and the computation time of a large-scale fast Fourier transform application with $\rho = 128$ (i.e., $|N| = 1151$) for varying reliability requirements. $R_{\text{seq}}(G)$ is also changed from 0.9 to 0.99 with 0.01 increments.

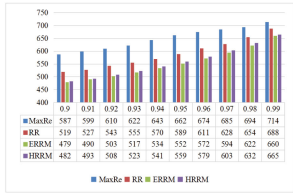
Fig. 5a shows the actual reliability values of the large-scale fast Fourier transform application on different reliability requirements. All the algorithms can satisfy the given reliability requirements in all cases. Similar to the results of the small-scale application in Fig. 4a, MaxRe still generates the maximum reliability values followed by RR, HRRM, and ERRM. Maximum differences between actual reliability and given reliability requirement are 0.0747 ($R_{\text{seq}}(G) = 0.9$) and 0.0184 ($R_{\text{seq}}(G) = 0.90$) for MaxRe and RR, respectively. On the contrary, in all cases the differences remain the minimum and close to application's reliability requirements using HRRM (0.0001-0.0003) and ERRM (0.0001-0.0002).

Fig. 5b shows the total number of replicas of the large-scale fast Fourier transform application on different reliability requirements. Similar to Fig. 4b in small-scale, MaxRe still generates the maximum numbers of replicas followed by RR, HRRM, and ERRM in all cases. The numbers of replicas using HRRM and ERRM are still very close and are much lower than those using MaxRe and RR in most cases.

Fig. 5c shows the computation time values of the large-scale fast Fourier transform application for reliability requirements. The values show that computation time is within 2.1 second using MaxRe, RR, and HRRM, whereas those using ERRM are 80-120 times longer. Such results indicate that ERRM is time-consuming for large-scale applications, as analyzed earlier.



(a) Actual reliability values.



(b) Total numbers of replicas.

Fig. 6. Results of the small-scale Gaussian elimination application on different reliability requirements (Experiment 3).

An interesting phenomenon is that the computation time values using ERRM for large scale applications are not increased but reduced as the application's reliability requirements increase in most cases, shown in Fig. 5c. The reason is that when using ERRM, it first calls the LBR algorithm (Algorithm 1) to obtain the initial reliability values of the application. A higher reliability requirement of the application may lead to higher initial reliability values with very short time by using LBR in these cases, such that the total computation time is not increased, but reduced with the application's reliability requirements increase.

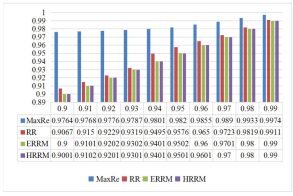
The results of Figs. 4a, and 5c show that ERRM and HRRM algorithms generate less redundancy than the state-of-the-art MaxRe and RR algorithms. Specifically, results of HRRM algorithm are very similar to those of ERRM algorithm indicating that HRRM implements approximate optimal redundancy with minimum time, whereas the enough optimal ERRM algorithm is time-consuming for large-scale parallel applications.

7.3 Gaussian Elimination Application

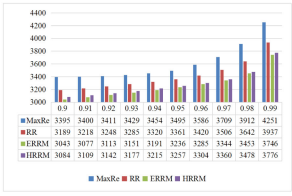
Experiment 3. This experiment compares the actual reliability values and the total number of replicas of in a small-scale Gaussian elimination application with $\rho = 21$ (i.e., $|N|$=230). The total number of task for the Gaussian elimination is similar to that of the fast Fourier transform application for varying reliability requirements. $R_{\text{seq}}(G)$ is also changed from 0.9 to 0.99 with 0.01 increments. Similar to small-scale fast Fourier transform in Experiment 1, the computation time values using all the algorithms are also within one second for the small-scale Gaussian elimination. Therefore, we also no longer list such values in this experiment.

Figs. 6a and 6b show the actual reliability values and total number of replicas of the small-scale Gaussian elimination application on different reliability requirements. In general, Experiment 3 shows similar pattern and values as Experiment 1 for the total number of replicas for all the algorithms.

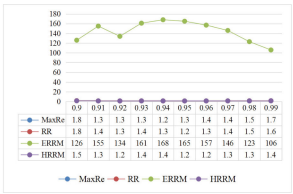
The results of Experiments 1 and 3 indicate that different parallelism degrees of applications in the same small-scale will generate similar actual reliability values and total number of replicas. In other words, parallelism degrees do not affect the scopes of actual reliability values and total number of replicas. The reason is that the reliability value of the application is the product of that of each task according to Eq. (4); considering that the number of tasks, the reliability requirement, and computation time are approximate equal, the actual reliability values and total number of replicas are also approximate equal.



(a) Actual reliability values.



(b) Total numbers of replicas.



(c) Computation time (unit: second).

Fig. 7. Results of the large-scale Gaussian elimination application on different reliability requirements (Experiment 4).

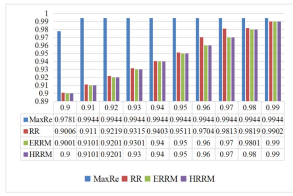
Experiment 4. This experiment compares the actual reliability values, the total number of replicas, and the computation time of a large-scale Gaussian elimination application with $\rho = 47$ (i.e., $|N| = 1,127$) for varying reliability requirements. $R_{\text{seq}}(G)$ is also changed from 0.9 to 0.99 with 0.01 increments.

Figs. 7a, 7b, and 7c show the actual reliability values, total number of replicas, and computation time of the large-scale Gaussian elimination application on different reliability requirements. Experiment 4 shows similar pattern and values as Experiment 2 in actual reliability values and total number of replicas for all the algorithms. The results of Experiments 2 and 4 further indicate that parallelism degrees do not affect the scopes of actual reliability values and total number of replicas.

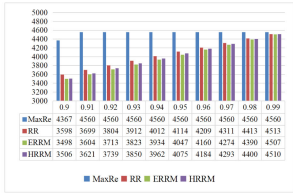
7.4 Randomly Generated Parallel Application

To extensively demonstrate the benefits of the proposed algorithms, we consider randomly generated parallel applications by the task graph generator [40]. Considering that the objective platform is heterogeneous processors, heterogeneity degrees may also affect the redundancy of application. Heterogeneity degree is easy to be implemented for randomly generated parallel applications as long as adjust the heterogeneity factor values. Randomly generated parallel applications are generated depending on the following parameters: average computation time is 50,000 ms, communication to computation ratio (CCR) is 1, and shape parameter is 1. The heterogeneity degree (factor) values belong to the scope of (0,1] in the task graph generator, where 0 and 1 represent the lowest and highest heterogeneity factors, respectively. Without loss of generality, we use large-scale randomly generated parallel application with 1,140 tasks, which are approximate equal to those of fast Fourier transform and Gaussian elimination applications in Experiments 2 and 4.

Experiment 5. This experiment compares the actual reliability values and the total number of replicas of a large-scale low-heterogeneity (with the heterogeneity factor 0.1) randomly generated parallel application with $|N| = 1,140$ for varying reliability requirements. $R_{\text{seq}}(G)$ is also changed from 0.9 to 0.99 with 0.01 increments.



(a) Actual reliability values.



(b) Total numbers of replicas.

Fig. 8. Results of the large-scale low-heterogeneity randomly generated parallel application on different reliability requirements (Experiment 5).

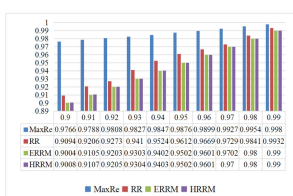
Figs. 8a and 8b show the actual reliability values and total numbers of replicas the large-scale low-heterogeneity randomly generated parallel application on different reliability requirements. It is easy to see that Experiment 5 shows similar pattern and values as Experiments 2 and 4 using all the algorithms. The main differences are as follows:

(1) The actual reliability values and total numbers of replicas obtained by MaxRe in Experiment 5 are relatively stable for different reliability requirements. The reason is that the execution time values are relative stable on the same processor for a low-heterogeneity parallel application and the reliability requirement using MaxRe is the same for all tasks, such that the values for the application do not changed much.

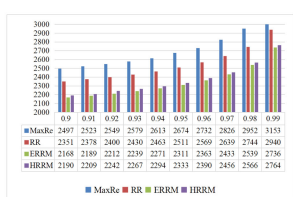
(2) The actual reliability values and total numbers of replicas obtained by RR, ERRM, and HRRM are relatively close in the same reliability requirement. The reason is still that the execution time values are relative stable on the same processor for a low-heterogeneity parallel application.

Experiment 6. This experiment compares the actual reliability values and the total number of replicas of a large-scale high-heterogeneity (with the heterogeneity factor 1) randomly generated parallel application with $|N| = 1140$ for varying reliability requirements. $R_{\text{seq}}(G)$ is also changed from 0.9 to 0.99 with 0.01 increments.

Figs. 9a and 9b show the actual reliability values and total numbers of replicas the large-scale high-heterogeneity randomly generated parallel application on different reliability requirements. It is easy to see that Experiment 6 shows similar pattern and values as Experiment 5 using all the algorithms. The main difference is that the high-heterogeneity application needs fewer replicas than the low-heterogeneity application. The total numbers of replicas for the former is only 60 percent of those for the latter using all the algorithms. The reason is that the actual reliability values for a task on different processors change much in a high-heterogeneity application, and these algorithms tend to choose the processor



(a) Actual reliability values.



(b) Total numbers of replicas.

Fig. 9. Results of the large-scale high-heterogeneity randomly generated parallel application on different reliability requirements (Experiment 6).

with the maximum reliability value for each task replication. Moreover, different from the low-heterogeneity application where the total numbers of replicas obtained by RR, ERRM, and HRRM are relatively close, ERRM and HRRM generate much less replicas than RR for the high-heterogeneity application. The reason is still that the actual reliability values for a task on different processors change much.

7.5 Summary of Experiments

Based on the above experimental results, summarizations are as follows.

(1) The proposed redundancy minimization algorithms, ERRM and HRRM, can generate less redundancy than the state-of-the-art MaxRe and RR algorithm at different scales, parallelism degrees, and heterogeneity degrees.
(2) Results of the HRRM algorithm are very similar to those of the ERRM algorithm. HRRM implements approximate optimal redundancy with minimum computation time, whereas the enough optimal ERRM algorithm is time-consuming for large-scale parallel applications.
(3) According to the analysis of the number of active processors, parallelism degrees do not affect the scopes of reliability values and total number of replicas for different types of applications in the same-scale.
(4) If the parallel application is small, then ERRM can be utilized to minimize redundancy; otherwise HRRM is the preferred alternative for reducing redundancy with minimum computation time.
(5) RR, ERRM, and HRRM obtain relatively close numbers of replicas for a low-heterogeneity application, whereas ERRM and HRRM obtain much less replicas than RR for the high-heterogeneity application. In other words, ERRM and HRRM are better suitable for high-heterogeneity applications than for low-heterogeneity applications.

8 Conclusion

We developed enough and heuristic replication algorithms ERRM and HRRM to minimize the redundancy for a parallel application in heterogeneous service-oriented systems. The ERRM algorithm can enough minimize redundancy by presenting two-stage replications. To decrease the time complexity of the time-consuming ERRM algorithm, the HRRM algorithm was also presented to deal with large-scale parallel applications within a short time. The main advantage for HRRM is its capability to obtain lower sub-reliability requirements for most tasks compared with MaxRe and RR, such that HRRM can generate less redundancy than MaxRe and RR. Results of our experiments on real and random generaed parallel applications at different scales, parallelism degrees, and heterogeneity degrees validate that both ERRM and HRRM generate less redundancy than the state-of-the-art MaxRe and RR algorithms. Experiment results also show that the HRRM implements approximate optimal redundancy with a short computation time. We believe that the proposed algorithms can effectively facilitate a reliability-aware design for parallel applications in heterogeneous service-oriented systems.

Resource usage and shortest schedule length are also important concern in high-performance computing systems. In fact, minimum redundancy does not mean minimum resource usage and shortest schedule length for a parallel application on heterogeneous systems because the same task has different execution time values on different processors. In our future work, we will consider the resource usage and schedule length minimization in such environment.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers for their constructive comments which have helped to improve the quality of the paper. This work was partially supported by the National Key Research and Development Plan of China under Grant Nos. 2016YFB0200405 and 2012AA01A301-01, the Natural Science Foundation of China under Grant Nos. 61672217, 61173036, 61432005, 61370095, 61300037, 61502405, 61300039, 61402170, and 61502162, the China Postdoctoral Science Foundation under Grant No. 2016M592422.

Supplement Material

The web page http://esnl.hnu.edu.cn/index.php/tsc/ publishes the experimental codes of the paper.

References

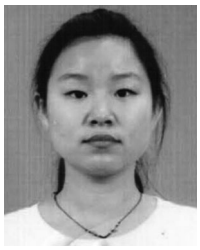
[1] Z. Cai, X. Li, and J. N. D. Gupta, "Heuristics for provisioning services to workflows in XaaS clouds," *IEEE Trans. Services Comput.*, vol. 9, no. 2, pp. 250–263, Mar./Apr. 2016.
[2] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. Chang, M. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Services Comput.*, vol. PP, no. 99, p. 1, Jan. 2016, doi: 10.1109/TSC.2016.2519898.
[3] Z. Fu, F. Huang, X. Sun, A. Vasilakos, and C.-N. Yang, "Enabling semantic search based on conceptual graphs over encrypted outsourced data," *IEEE Trans. Services Comput.*, vol. PP, no. 99, p. 1, Oct. 2016, doi: 10.1109/TSC.2016.2622697.
[4] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Feb. 2016.
[5] Y. Kong, M. Zhang, and D. Ye, "A belief propagation-based method for task allocation in open and dynamic cloud environments," *Knowl.-Based Syst.*, vol. 115, pp. 123–132, Jan. 2017.
[6] F. Zhangjie, S. Xingming, L. Qi, Z. Lu, and S. Jiangang, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Trans. Commun.*, vol. 98, no. 1, pp. 190–200, Jan. 2015.
[7] Q. Liu, W. Cai, J. Shen, Z. Fu, X. Liu, and N. Linge, "A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment," *Secur. Commun. Netw.*, vol. 9, no. 17, pp. 4002–4012, Nov. 2016.
[8] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment," *J. Grid Comput.*, vol. 14, no. 1, pp. 55–74, Mar. 2016.
[9] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Aug. 2002.
[10] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Comput.*, vol. 38, no. 4, pp. 175–193, Apr. 2012.
[11] G. Xie, R. Li, and K. Li, "Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems," *J. Parallel Distrib. Comput.*, vol. 83, pp. 1–12, May. 2015.

[12] G. Xie, X. Xiao, R. Li, and K. Li, "Schedule length minimization of parallel applications with energy consumption constraints using heuristics on heterogeneous distributed systems," *Concurrency Comput.-Parctice Experience*, pp. 1–10, Nov. 2016, doi: 10.1002/cpe.4024.
[13] J.-S. Leu, C.-F. Chen, and K.-C. Hsu, "Improving heterogeneous SOA-Based iot message stability by shortest processing time scheduling," *IEEE Trans. Services Comput.*, vol. 7, no. 4, pp. 575–585, Oct.-Dec. 2014.
[14] L. Zhao, Y. Ren, Y. Xiang, and K. Sakurai, "Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems," in *Proc. 12th IEEE Int. Conf. High Perform. Comput. Commun.*, 2010, pp. 434–441.
[15] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Comput.*, vol. 39, no. 10, pp. 567–585, Jul. 2013.
[16] Z. Zheng, T. C. Zhou, M. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Trans. Services Comput.*, vol. 5, no. 4, pp. 540–550, Oct.-Dec. 2012.
[17] W. Qiu, Z. Zheng, X. Wang, X. Yang, and M. R. Lyu, "Reliability-based design optimization for cloud migration," *IEEE Trans. Services Comput.*, vol. 7, no. 2, pp. 223–236, Apr.-Jun. 2014.
[18] M. Silic, G. Delac, and S. Srbljic, "Prediction of atomic web services reliability for QoS-Aware recommendation," *IEEE Trans. Services Comput.*, vol. 8, no. 3, pp. 425–438, May/Jun. 2015.
[19] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 241–254, Oct.-Dec. 2009.
[20] A. Benoit, M. Hakem, and Y. Robert, "Fault tolerant scheduling of precedence task graphs on heterogeneous platforms," in *Proc. 22th IEEE Int. Parallel Distrib. Process.*, 2008, pp. 1–8.
[21] A. Benoit and M. Hakem, "Optimizing the latency of streaming applications under throughput and reliability constraints," in *Proc. 45th Int. Conf. Parallel Process.*, 2009, pp. 325–332.
[22] [Online]. Available: https://en.wikipedia.org/wiki/IEC_61508
[23] [Online]. Available: https://en.wikipedia.org/wiki/ISO_9000
[24] G. Xie, L. Liu, L. Yang, and R. Li, "Scheduling trade-off of dynamic multiple parallel workflows on heterogeneous distributed computing systems," *Concurrency Comput.-Parctice Experience*, vol. 29, no. 2, pp. 1–18, Jan. 2017, doi: 10.1002/cpe.3782.
[25] V. T'kindt and J.-C. Billaut, *Multicriteria Scheduling: Theory, Models and Algorithms*. Berlin, Germany: Springer, Mar. 2006.
[26] A. Doğan and F. Özgüner, "Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems," *Comput. J.*, vol. 48, no. 3, pp. 300–314, Mar. 2005.
[27] M. Hakem and F. Butelle, "A bi-objective algorithm for scheduling parallel applications on heterogeneous systems subject to failures," in *Proc. RenPar2006*, 2006, pp. 25–35.
[28] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proc. 19th ACM Int. Symp. Parallel Algorithms Architectures*, 2007, pp. 280–288.
[29] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented grids and utility computing: The state-of-the-art and future directions," *J. Grid Comput.*, vol. 6, no. 3, pp. 255–276, Sep. 2008.
[30] [Online]. Available: https://en.wikipedia.org/wiki/Service-level_agreement
[31] S. M. Shatz and J. P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Rel.*, vol. 38, no. 1, pp. 16–27, Apr. 1989.
[32] J. Mei, K. Li, X. Zhou, and K. Li, "Fault-tolerant dynamic rescheduling for heterogeneous computing systems," *J. Grid Comput.*, vol. 13, no. 4, pp. 507–525, Dec. 2015.
[33] X. Qin, H. Jiang, and D. R. Swanson, "An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems," in *Proc. 31th Int. Conf. Parallel Process.*, 2002, pp. 360–368.
[34] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Comput.*, vol. 32, no. 5, pp. 331–356, Jun. 2006.
[35] Q. Zheng, B. Veeravalli, and C.-K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 380–393, Mar. 2009.
[36] A. Benoit, L.-C. Canon, E. Jeannot, and Y. Robert, "Reliability of task graph schedules with transient and fail-stop failures: Complexity and algorithms," *J. Scheduling*, vol. 15, no. 5, pp. 615–627, Oct. 2012.
[37] A. Verma and N. Bhardwaj, "A review on routing information protocol (RIP) and open shortest path first (OSPF) routing protocol," *Int. J. Future Generation Commun. Netw.*, vol. 9, no. 4, pp. 161–170, Apr. 2016.
[38] Q. Zheng and B. Veeravalli, "On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices," *J. Parallel Distrib. Comput.*, vol. 69, no. 3, pp. 282–294, 2009.
[39] L. Zhao, Y. Ren, and K. Sakurai, "A resource minimizing scheduling algorithm with ensuring the deadline and reliability in heterogeneous systems," in *Proc. 25th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2011, pp. 275–282.
[40] [Online]. Available: https://sourceforge.net/projects/taskgraphgen/

Guoqi Xie received the PhD degree in computer science and engineering from Hunan University, China, in 2014. He was a postdoctoral researcher with Nagoya University, Japan, from 2014 to 2015. Since 2015, he is working as a postdoctoral researcher with Hunan University, China. He has received the best paper award from ISPA 2016. His major interests include embedded and real-time systems, parallel and distributed systems, software engineering and methodology. He is a member of the IEEE, the ACM, and the CCF.

Gang Zeng received the PhD degree in information science from Chiba University, in 2006. He is an associate professor in the Graduate School of Engineering, Nagoya University. From 2006 to 2010, he was a researcher, and then assistant professor in the Center for Embedded Computing Systems (NCES), Graduate School of Information Science, Nagoya University. His research interests mainly include power-aware computing and real-time embedded system design. He is a member of the IEEE and the IPSJ.

Yuekun Chen is currently working toward the PhD degree at Hunan University. Her research interests include services and cloud computing, fault-tolerance computing, and software engineering.



Yang Bai is currently working toward the PhD degree at Hunan Province, Hunan University. Her research interests include service computing, embedded systems, and cyber-physical systems.

Zhili Zhou is an assistant professor with Nanjing University of Information Science and Technology. His research interests include cloud computing, information forensics and security, pattern recognition.

Renfa Li is a professor of computer science and electronic engineering, and the dean of the College of Computer Science and Electronic Engineering, Hunan University, China. He is the director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. His major interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of things. He is a member of the council of CCF, a senior member of the IEEE, and a senior member of the ACM.

Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things, and cyber-physical systems. He has published more than 460 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.

Mining Deficiencies of Online Reputation Systems: Methodologies, Experiments and Implications

Hong Xie, *Member, IEEE* and John C.S. Lui, *Fellow, IEEE*

Abstract—Online reputation systems serve as core building blocks in various Internet services such as E-commerce (e.g., eBay) and crowdsourcing (e.g., oDesk). The flaws and deficiencies of real-world online reputation systems have been reported extensively. Users who are frustrated about the system will eventually abandon such service. However, there is no systematic and formal studies which examine such deficiencies. This paper presents the first attempt, which develops a novel data analytical framework to uncover online reputation system deficiencies from data. We develop two novel measures to quantify the efficiency of online reputation systems: (1) ramp up time of a new service provider, (2) long term profit gains for a service provider. We present a new data analytical framework to evaluate these two measures from data. We show that inherent preferences or personal biases in expressing feedbacks (or ratings) cause the computational infeasibility in evaluating the ramp up time and the long term profit gains from data. We develop two computationally efficient randomized algorithms with theoretical performance guarantees to address this computational challenge. We apply our methodology to analyze real-life datasets (from eBay, Google Helpouts, Amazon and TripAdvisor). We extensively validate our model and we uncover the deficiencies of online reputation systems. Our experimental results uncovers insights on why Google Helpouts was eventually shut down in April 2015 and why eBay is losing some sellers heavily.

Index Terms—Online reputation systems, ramp up time, long term profit gains, approximation algorithms

1 Introduction

With the advancement of Internet technologies, a variety of online services are booming. E-commerce systems such as eBay [7] and Taobao [25] of Alibaba are representative examples. In an E-commerce system, buyers can purchase products from strangers and transactions are conducted online. Another typical Internet service is online product review website such as TripAdvisor [27], etc. In such websites, customers share their experiences on products, so that other customers can make purchasing decisions based on these experiences. Crowdsourcing services such as Google Helpouts [9] and oDesk [20] are another form of Internet services, where requester can outsource a task to different workers. One common characteristic of the above services is that transactions are usually carried out between two "strangers", and there is a risk because sellers may sell low quality goods while workers may provide low quality solutions. To overcome such risk, Internet service companies deploy reputation systems [21].

In general, an online reputation system involves two parties: "*service providers*" and "*customers*". A service provider can be a seller in eBay, a worker in Google Helpouts, or a hotel chain in TripAdvisor. A customer can be a buyer in eBay, a task requester in Google Helpouts, or a traveller in TripAdvisor. A transaction can be a buyer purchasing a product from a seller, a requester paying a worker to solve a task, or a customer spending an evening in a hotel. When a transaction is completed, a customer gives a feedback rating to indicate the quality of a service. For example, eBay adopts a three-level cardinal rating metric: {"negative", "neutral", "positive"}. Each service provider is associated with a reputation score, which is the aggregation of all its feedback ratings. The reputation score reflects the "*overall quality*" of service providers, and each service provider's reputation is accessible by all customers.

Many reports have indicated that existing online reputation systems have critical flaws, which result in losing users and putting Internet service companies at the risk of significant revenue loss. For example, it was reported in [29] that the eBay reputation system frustrates sellers. More concretely, the eBay reputation system forces some sellers out of the business because it makes them difficult to attract customers. In fact, eBay was reported to have a significant user loss [12], [23], [26]. Similarly, Google Helpouts was eventually shut down in April 2015 due to poor business [8]. It is important to formally explore these phenomena: *What are the key factors which influence the efficiency of online reputation systems? How to uncover the deficiencies of online reputation systems from data?* Exploring these questions not only can help us to uncover potential risks of online reputation systems, but we also can gain important insights to improve them.

• *The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong. E-mail: hongx87@gmail.com, cslui@cse.cuhk.edu.hk.*

Manuscript received 2 Jan. 2017; revised 10 June 2017; accepted 10 July 2017. Date of publication 21 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Hong Xie.)
Digital Object Identifier no. 10.1109/TSC.2017.2730206

Despite its importance, there is no formal study to explore deficiencies of online reputation systems. This paper aims to fill this void. However, there exists at least three challenges: (1) What is the right performance measure to quantify the *efficiency* of online reputation systems? (2) How to efficiently process online reputation datasets and apply these measures to analyze them? (3) How to address computational challenges arose in large scale reputation data analysis? This paper addresses these challenges. Our contributions are:

- We propose two measures: *ramp up time* and *long term profit gains*, to quantify the efficiency of feedback-based online reputation systems.
- We show that preferences or personal biases in assigning feedbacks (or ratings) cause the computational infeasibility in evaluating our proposed measures from data. We propose computationally efficient randomized algorithms (with theoretical performance guarantees) to address the above computational challenge.
- We apply our methodology to real-life datasets from eBay, Google Helpouts, Amazon and TripAdvisor. We extensively validate our model. We discover the deficiencies of online reputation systems: (1) the ramp up time is more than 500 days; (2) reducing ramp up time can improve the long term profit gains significantly, e.g., an 80 percent reduction on ramp up time leads to at least 50 percent (as high as 100 percent) improvement in long term profit gains. Our experimental results also uncovers insights on why Google Helpouts was eventually shut down in April 2015 and why eBay is losing sellers heavily.

This paper organizes as follows. In Section 2, we present the system model for online reputation systems. In Section 3 we formulate our problem. In Section 4, we present a novel data analytical framework to evaluate reputation systems. In Section 5 We then extend our framework to incorporate preferences or personal biases in assigning feedbacks (or ratings). In Section 6 we present the design of two randomized algorithms to approximate ramp up time and long term profit gains respectively. In Section 7 we present experimental results using datasets from eBay and Google Helpouts. Related work is given in Section 8 and we conclude in Section 9.

2 System Model

We present a general model to characterize online reputation systems, which are deployed in different types of Internet services, e.g., electronic commerce like eBay [7], crowdsourcing services like Google Helpouts [9], and hotel review websites like TripAdvisor [27]. In general, such Internet services consist of "*service providers*", "*customers*" and a "*reputation system*".

- *Service Providers:* we define "service providers" as users that supply items. A service provider can be a seller in eBay, a worker in Google Helpouts, or a hotel chain in TripAdvisor.
- *Customers:* we define "customers" as users that purchase items. A customer can be a buyer in eBay, a task requester in Google Helpouts, or a traveller in TripAdvisor.

TABLE 1
Main Notations

g	unit per product profit gain
$r(t)$	the average rating up to time slot t
$\mathcal{R}(\cdot)$	the rating mapping function
m	the total number of rating levels
$n_i(t)$	the number of level i rating up to time slot t
$Q, \widehat{Q}$	the intrinsic, perceived quality of a service provider
γ, N_h	threshold on average rating, total number of ratings
λ_1, λ_2	transaction rate before, after ramping up
$\lambda(t)$	transaction rate at time slot t
$T_t, E[T_r]$	ramp up time, expected ramp up time
G	expected long term profit gains
$\widehat{E}[T_r], \widehat{G}$	estimated ramp up time, long term profit gains
η_i	the probability of receiving a level i rating
δ	the discounting factor
ϵ	the relative estimation error
ξ	the fail probaibility

Customers conduct transactions with service providers. To encourage transactions among customers and service providers, Internet service companies use reputation systems to reflect the "*overall quality*" of service providers, and each service provider's reputation is accessible by all customers. Clearly, a service provider having a high reputation can attract more customers, which leads to larger revenue. Table 1 lists all the key notations.

2.1 Transaction Model

A customer pays a fee, which we call the $price$, to a service provider in order to complete a transaction. For example, a buyer pays a seller some money to buy a product, or a requester pays a worker some money to have a task solved. Without loss of generality, we focus on a normalized price, i.e., $price \in [0, 1]$. A service provider incurs a $cost \in [0, 1]$, in providing a service to a customer, i.e., a product has a manufacturing cost, or solving a task has a cost. The internet service company (e.g., eBay, Taobao, Google Helpouts), charges a $transaction\ fee \in [0, 1]$ for each transaction. Our analysis also applies for a transaction fee, which is proportional to the price, because the focus of this paper is not on the pricing strategies. Thus, for brevity, we consider a fixed transaction fee. A service provider receives a unit profit gain of g for a completed transaction

$$g \triangleq price - cost - transaction\ fee. \tag{1}$$

To incentivize service providers to participate, we must have $g > 0$, which yields $transaction\ fee < price - cost$.

2.2 Model for Online Reputation Systems

Many Internet service companies deploy reputation systems to reflect the overall quality of service providers. For example, eBay maintains a reputation system to reflect the trustworthiness of sellers. In general, such reputation systems are composed of a "*feedback rating mechanism*" and "*a rating aggregating policy*".

When a transaction completes, a customer expresses a feedback rating to indicate the quality of a service. One most commonly adopted rating metric is the m-level cardinal rating metric $\{1, \ldots, m\}$, where $m \geq 2$. For example, eBay adopts a three-level cardinal rating metric: {1 =

"negative", 2 = "neutral", 3 = "positive"}, while TripAdvisor adopts a five-level cardinal rating metric: {1="Terrible", 2 = "Poor", 3 ="Average", 4="Very good", 5="Excellent"}. Each rating level is associated with a numerical score, which is used to compute reputation scores for a service provider. Let $\mathcal{R} : \{1, \ldots, m\} \to \mathbb{R}$ denote a map which prescribes a score for each rating level, i.e., eBay adopts $\mathcal{R}(1) = -1, \mathcal{R}(2) = 0, \mathcal{R}(3) = 1$, and TripAdvisor adopts $\mathcal{R}(i) = i, \forall i = 1, \ldots, 5$.

Each service provider is associated with a reputation score, which is an "*aggregation*" of all its feedback ratings. The reputation score quantifies the overall reputation of a service provider. One most widely adopted rating aggregating rule is the average score rule. Let r denote the reputation score of a service provider. Let n_i denote the number of ratings that are of rating level i, where $i = 1, \ldots, m$, we have

$$r = \frac{\sum_{i=1}^{m} n_i \mathcal{R}(i)}{\sum_{i=1}^{m} n_i}. \tag{2}$$

The reputation score r is a public information accessible by all customers. For the ease of presentation, this paper focuses on the average score rule. We will see later, the results can be extended to the weighted average score rule.

We now describe the reputation updating process. Let $(r, n_1, \ldots, n_m)$ be the reputation profile for a service provider. In order to assist customers to assess the overall reputation of service providers, Internet service companies publish reputation profiles to the public. We use a discrete time system to characterize the reputation updating process. Let $(r(t), n_1(t), \ldots, n_m(t))$ be the reputation profile of a service provider at time slot $t \in \{0, 1, \ldots, \infty\}$, where $r(t)$ is its reputation score up to time slot t, and $n_i(t)$ is the cumulative number of level i ratings up to time slot t. Each service provider is initialized with $(0, 0, \ldots, 0)$. Let $N_i(t)$ denotes the number of transactions completed in time slot t that lead to the level i rating. In real-world reputation systems, reputation updating has delays. We assume that the delay is one time slot. The reputation profile is updated as follows:

$$\begin{cases} r(t+1) = \frac{r(t)\sum_{i=1}^{m} n_i(t) + \sum_{i=1}^{m} \mathcal{R}(i) N_i(t)}{\sum_{i=1}^{m} n_i(t) + \sum_{i=1}^{m} N_i(t)}, \\ n_i(t+1) = n_i(t) + N_i(t), \text{for all } i = 1, \ldots, m. \end{cases} \tag{3}$$

For brevity, we drop the time stamp t in our analysis when there is no confusion.

2.3 Model for Rating Behavior

Each service provider has an intrinsic quality, which indicates his true overall service quality. For example, a high quality seller in eBay sells high quality products and provides fast shipment. Let $Q \in [\mathcal{R}(1), \mathcal{R}(m)]$ denote the intrinsic quality of a service provider. After completing a transaction, a customer perceives the quality of a service provider, which is denoted by $\hat{Q} \in [\mathcal{R}(1), \mathcal{R}(m)]$. Customers pick the rating level, which is the most accurate one in reflecting the perceived quality $\hat{Q}$. Formally, we have

$$\text{feedback rating} = \arg\min_{i \in \{1, \ldots, m\}} \left| \mathcal{R}(i) - \hat{Q} \right|, \tag{4}$$

where the feedback rating denotes the individual rating assigned by a customer. For example, consider $m = 5$ and $\mathcal{R}(i) = i$ for all $i \in \{1, \ldots, 5\}$. When $\hat{Q} = 4.7$, the feedback rating will be 5. When $\hat{Q} = 4.4$, the feedback rating will be 4. Consider $\hat{Q} = 4.5$, then according to Equation (4), both rating 4 and 5 are valid. In such ties, we pick the larger one, e.g., here we pick 5 to model that customers are lenient in assigning ratings. This choice will not influence the results for this paper.

For the purpose of illustrating intuitions and key ideas, we first assume that there are no errors in perceiving quality, i.e., $\hat{Q} = Q$. We will extend our model to accommodate quality perceiving errors (i.e., $\hat{Q} \neq Q$) in Section 5.

2.4 Model for Transaction's Arrival Rate

We now quantify the impact of a reputation system on service providers' revenue. The reputation system builds trust among customers and service providers. This trust is critical in attracting transactions. More precisely, customers aim to minimize the risk in service purchase and they prefer to interact with reputable service providers.

Based on the reputation profile, we categorize service providers into two types: "*reputable*", and "*average*". Note that each service provider is initialized with $(0, 0, \ldots, 0)$. To earn a reputable label, a service provider must improve his reputation to meet two requirements. The first one is that the reputation score r must be larger than or equal to a threshold $\gamma \in [\mathcal{R}(1), \mathcal{R}(m)]$. This requirement shows that a service provider can provide services with a high overall quality. The second requirement is that the number of feedback ratings must be larger than or equal to a threshold N_h. This requirement guarantees that the reputation score is statistically significant. Otherwise, a service provider is labeled as average.

Definition 1. *A reputable service provider must satisfy the following two conditions: $r \geq \gamma$ and $\sum_{i=1}^{m} n_i \geq N_h$. A service provider is labeled as an average service provider if and only if $r < \gamma$ or $\sum_{i=1}^{m} n_i < N_h$.*

Note that a new service provider is initialized with $(0, 0, \ldots, 0)$. Hence, a new service provider is always labeled as "*an average service provider*". We need both the rating scale and score scale in order to make our model practical, because in real-world applications such as eBay and Google helpouts both of these two scales are be displayed to users. These two scales are two important indicators of a service provider's reputation.

A service provider's label (i.e., reputable, or average label) is critical to its revenue. Customers are more willing (unwilling) to conduct transactions with reputable (average) service providers. Let λ_1 and λ_2 be the transaction's arrival rate when a service provider is labeled as average and reputable respectively. The transaction's arrival rate satisfies $\lambda_1 < \lambda_2$, which signifies that a reputable service provider can attract more transactions.

Definition 2. *Denote $\lambda(t)$ the transaction's arrival rate at time slot t. Formally we can express it as*

$$\lambda(t) = \begin{cases} \lambda_1, & \text{if } r(t) < \gamma \text{ or } \sum_{i=1}^{m} n_i(t) < N_h, \\ \lambda_2, & \text{if } r(t) \geq \gamma \text{ and } \sum_{i=1}^{m} n_i(t) \geq N_h. \end{cases} \tag{5}$$

Let $N(t)$ denote the number of transactions that arrive to a service provider in time slot t. Then we have $E[N(t)] = \lambda(t)$. This paper focuses on that $N(t)$ follows a Poisson distribution with parameter $\lambda(t)$. This point is verified on real-world dataset in Section 7.4.

Remark. Our model focuses on two types of rates, i.e., λ_1, λ_2, in order to strike a good balance between simplicity and practicability. It is important to keep the simplicity of our model, since it enables us to present the key ideas and insights in a clear fashion. Note that our model is practical enough as well. To uncover the deficiencies a reputation system, it is reasonable to examine the average rate to service providers (either average or reputable provider). The transactions' rate λ_1 and λ_2 can be interpreted as the average rate to average service providers and reputable service providers respectively. These two types of rates are sufficient to capture the key impact of a reputation system on the profit of service providers.

3 Problem Statement

We formulate two novel measures to quantify the efficiency of online reputation systems: (1) *ramp up time* T_r, (2) *expected long term profit gains* G for a service provider. We pose a question of how to infer these two measures from real-world online reputation systems' datasets and how to uncover inefficiencies and limitations of real-world reputation systems.

3.1 Ramp Up Time

The number of time slots that a new service provider needs so to ramp up his reputation (or attain a "reputable" label) is critical to his revenue and it also affects the transaction gains of the Internet service company. Recall that each new service provider is initialized with an average label, while customers are more willing to conduct transactions with reputable service providers than average labeled service providers. Hence, it is critical for a service provider to earn a reputable label quickly so as to increase the transaction volume. Furthermore, the Internet service company will have a higher transaction gain when transaction volume increases. We next state the ramp up condition and the ramp up process.

Definition 3. *A new service provider's reputation profile is* $(0, 0, \ldots, 0)$. *To earn a reputable label, he must collect enough high feedback ratings. We define the process of earning a reputable label, i.e., increasing his reputation to* $r \geq \gamma$ *and* $\sum_{i=1}^{m} n_i \geq N_h$, *as the ramp up process. Furthermore, we say that a service provider satisfies the ramp up condition iff* $r \geq \gamma$ *and* $\sum_{i=1}^{m} n_i \geq N_h$.

Recall that a service provider's reputation profile at time slot t is $(r(t), n_1(t), \ldots, n_m(t))$. In the following, we formally define the ramp up time.

Definition 4. *The ramp up time is the minimum number of time slots that a service provider must spend to earn a reputable label. Let* T_r *denote the ramp up time*

$$T_r \triangleq \arg\min_t \left\{ r(t) \geq \gamma \text{ and } \sum_{i=1}^{m} n_i(t) \geq N_h \right\}. \quad (6)$$

The ramp up time quantifies the minimum time that a service provider must spend to earn a reputable label. It reflects how difficult it is for a service provider to start a business. If the ramp up time is large, a service provider may drop out or change to some other Internet service companies. We therefore consider the following problem.

Problem 1. *How to infer the ramp up time from real-world reputation system datasets and how it influences the efficiency of real-world reputation systems?*

3.2 Long Term Profit Gains

Profit gains are critical to service providers. They serve as one important incentive for service providers to maintain their business and they are one of the key motivations for service providers to join an Internet service company. If service providers have large profit gains, this also implies that the Internet service company (e.g., eBay or Alibaba) will have higher profit. On the other hand, service providers may quit if there is only a small profit gain, and this may lead to losses to an Internet service company.

We now formally quantify profit gains. Recall that a service provider earns a unit profit gain of g for completing one transaction (refer to Equation (1)). Note that $N(t)$ is a random variable which has a Poisson distribution with parameter $\lambda(t)$, where $\lambda(t)$ is expressed in Equation (5). Hence, on average, a service provider earns a profit gain of $gN(t)$ in the time slot t. Using micro-econometric analysis, we use a discounted long term profit gain to quantify service providers' total profit gains in time slot $0, 1, \ldots, \infty$. Let $\delta \in (0, 1]$ be the discounting factor.

Definition 5. *Denote* G *the expected long term profit gains for a service provider. We can express it as have*

$$G \triangleq E\left[\sum_{t=0}^{\infty} \delta^t g N(t)\right]. \quad (7)$$

We consider the following problem.

Problem 2. *How to infer the long term profit gains* G *from real-world reputation system datasets and reveal its impact on the real-world reputation systems?*

4 Baseline Data Analytical Framework

We first develop a data analytical framework to uncover deficiencies of online reputation systems. We then develop theoretical foundations for such framework, i.e., derive analytical expressions for the ramp up time T_r and the expected long term profit gains G. This gives us important insights to develop efficient algorithms to evaluate T_r and G from data.

4.1 Data Analytical Framework

Our baseline data analytical framework consists of three steps. In the first step, we infer model parameters, i.e., $m, \gamma, \mathcal{R}, N_h, \lambda_1$ and λ_2, from data. In the second step, we input them into our model, and apply our model to evaluate system efficiency measures, ramp up time T_r and long term profit gains G. In the third step we empirically analyze the ramp up time T_r and expected long term profit gains G so as to uncover deficiencies of online reputation systems. We outline this baseline data analytical framework in Algorithm 1.

Algorithm 1. Baseline Data Analytical Framework

1: **Parameter inference.** Infer model parameters $m, \gamma, \mathcal{R}, N_h, \lambda_1$ and λ_2 from data.
2: **Quantifying system efficiency.** Evaluating the ramp up time T_r and expected long term profit gains G based on these inferred parameters.
3: **Uncover system deficiencies.** Uncover deficiencies of online reputation systems via empirical studies on T_r and G.

In the remaining of this section, we focus on step two of the above framework, i.e., quantifying the system efficiency (we will present the details of step 1 and step 3 of the above framework in Section 7). More specifically, we derive analytical expressions for the ramp up time T_r and expected long term profit gains G, assuming that model parameters $m, \gamma, \mathcal{R}, N_h, \lambda_1$ and λ_2 are given. Then we apply them to develop efficient algorithms to evaluate T_r and G from data.

4.2 Algorithm to Evaluate Ramp Up Time

Recall that customers can perceive the intrinsic quality of service providers, i.e., $\hat{Q} = Q$. Applying Equation (4), we obtain that a service provide will receive $\arg\min_{i\in\{1,\ldots,m\}} |\mathcal{R}(i) - Q|$ level ratings. This means that the reputation score of a service provider will be of $r = \mathcal{R}(\arg\min_{i\in\{1,\ldots,m\}} |\mathcal{R}(i) - Q|)$. By Definition 4, a service provider can get ramped up if and only if $r \geq \gamma$, which yields $\mathcal{R}(\arg\min_{i\in\{1,\ldots,m\}} |\mathcal{R}(i) - Q|) \geq \gamma$. We can then introduce the notation of intrinsically reputable and average service providers respectively.

Definition 6. *We say a service provider is intrinsically reputable if and only if his intrinsic quality satisfies* $\mathcal{R}(\arg\min_{i\in\{1,\ldots,m\}} |\mathcal{R}(i) - Q|) \geq \gamma$, *otherwise we say a service provider is intrinsically average.*

We express the analytical expression for ramp up time T_r in the following theorem. This will give us important insights to develop algorithms to evaluate T_r from data.

Theorem 1. *Consider an intrinsically average service provider, the ramp up time can be expressed as*

$$T_r = \infty. \tag{8}$$

Consider an intrinsically reputable service provider, the expected ramp up time can be expressed as

$$E[T_r] = \sum_{t=1}^{\infty} \sum_{k=0}^{N_h-1} e^{-(t-1)\lambda_1} \frac{((t-1)\lambda_1)^k}{k!}. \tag{9}$$

Remark. All proofs to lemmas and theorems are in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2017.2730206. An intrinsically average service provider never gets ramped up. As long as we can infer λ_1 and N_h from data, we can evaluate the ramp up time $E[T_r]$ by applying Equations (8) and (9). However, one technical issue is that we have to perform a summation of infinity number of terms in order to evaluate Equation (9). In the following theorem we address this issue via truncation. Let $\hat{E}[T_r]$ denote an estimation of $E[T_r]$.

Theorem 2. *Let* $\hat{t} \in \{1, \ldots, \infty\}$ *and let* $\epsilon > 0$ *denote the relative error. Suppose*

$$\hat{E}[T_r] = \sum_{t=1}^{\hat{t}} \sum_{k=0}^{N_h-1} e^{-(t-1)\lambda_1} \frac{((t-1)\lambda_1)^k}{k!}.$$

If $\hat{t} > 2\max\{(-\ln(1-e^{-0.5\lambda_1}) + \ln\frac{1}{\epsilon} - \ln\frac{N_h}{\lambda_1})/\lambda_1, 4\frac{N_h-1}{\lambda_1} + \frac{1}{2}\}$, *then* $|\hat{E}[T_r] - E[T_r]| \leq \epsilon E[T_r]$.

Remark. The above theorem states an closed-form accurate estimation of the ramp up time. And the estimation error ϵ can be arbitrarily small by selecting a large enough $\hat{t}$. It is interesting to observe that $\hat{t}$ increases linearly in $\ln 1/\epsilon$.

Based on Theorem 2, we develop an efficient algorithm to evaluate the ramp up time in Algorithm 2. The computational complexity of Algorithm 2 is $\Theta(\lceil 2\max\{(-\ln(1-e^{-0.5\lambda_1}) + \ln\frac{1}{\epsilon} - \ln\frac{N_h}{\lambda_1})/\lambda_1, 4\frac{N_h-1}{\lambda_1} + \frac{1}{2}\}\rceil N_h) = \Theta(N_h \ln\frac{1}{\epsilon})$. This implies that Algorithm 2 is highly efficient. To apply Algorithm 2, we need to infer model parameters $\lambda_1, N_h, \mathcal{R}$ and γ from data (we will infer them in Section 7).

Algorithm 2. Evaluating Ramp Up Time

Input: Model parameters $\lambda_1, N_h, \mathcal{R}$ and γ. Accuracy factor ϵ. Intrinsic quality Q.
Output: $\hat{E}[T_r]$

if $\mathcal{R}(\arg\min_{i\in\{1,\ldots,m\}} |i - Q|) < \gamma$ **then**
  $\hat{E}[T_r] = \infty$
else
  $\hat{E}[T_r] \leftarrow 0$.
  $\hat{t} \leftarrow \lceil 2\max\{(-\ln(1-e^{-0.5\lambda_1}) + \ln\frac{1}{\epsilon} - \ln\frac{N_h}{\lambda_1})/\lambda_1, 4\frac{N_h-1}{\lambda_1} + \frac{1}{2}\}\rceil$.
  for $t = 1$ to $\hat{t}$ **do**
    for $k = 0$ to $N_h - 1$ **do**
      $\hat{E}[T_r] \leftarrow \hat{E}[T_r] + e^{-(t-1)\lambda_1} \frac{((t-1)\lambda_1)^k}{k!}$
    end for
  end for.
end if

4.3 Algorithm to Evaluate Long Term Profit Gains

To gain some insights in evaluating long term profit gains from data, we first close-form expression for them.

Theorem 3. *Consider an intrinsically average service provider, the long term profit gains can be expressed as*

$$G = \frac{g\lambda_1}{1-\delta}. \tag{10}$$

Consider an intrinsically reputable service provider, the expected long term profit gains can be expressed as

$$G = \frac{g\lambda_2}{1-\delta} + (\lambda_1 - \lambda_2) g \sum_{t=0}^{\infty} \sum_{k=0}^{N_h-1} \delta^t e^{-\lambda_1 t} \frac{(\lambda_1 t)^k}{k!}. \tag{11}$$

Remark. The implication of this theorem is that as long as we can infer λ_1, λ_2 and N_h from data, we can characterize the long term profit gains G by applying Equations (10) and (11). However, to compute G for intrinsically reputable service providers (i.e., Equation (11)) we have to perform a summation of infinity number of terms. In the following theorem we address this issue via truncation. Let $\hat{G}$ denote an estimation on the long term profit gains.

Theorem 4. *Denote $\hat{t} \in \{1, \ldots, \infty\}$. Consider an intrinsically reputable service provider, i.e., $\mathcal{R}(\arg\min_{i\in\{1,\ldots,m\}}|i-Q|) \geq \gamma$. Suppose*

$$\hat{G} = \frac{g\lambda_2}{1-\delta} + (\lambda_1 - \lambda_2)g \sum_{t=0}^{\hat{t}} \sum_{k=0}^{N_h-1} \delta^t e^{-\lambda_1 t} \frac{(\lambda_1 t)^k}{k!},$$

If $\hat{t} > \max\{(\ln\frac{1-\delta e^{-0.5\lambda_1}}{1-\delta} + \ln\frac{\lambda_1}{\lambda_2-\lambda_1} + \ln\epsilon)/(\ln\delta - 0.5\lambda_1) - 1, 8\frac{N_h-1}{\lambda_1}\}$, then $|\hat{G} - G| \leq \epsilon G$.

Remark. The above theorem states a closed-form accurate estimation of the long term profit gains. And the estimation error ϵ can be arbitrarily small by selecting a large enough $\hat{t}$. It is interesting to observe that $\hat{t}$ increases linearly in $\ln 1/\epsilon$.

Based on Theorem 4, we outline an algorithm to evaluate the ramp up time in Algorithm 3. The computationally complexity of Algorithm 3 is $\Theta(\lceil\max\{(\ln\frac{1-\delta e^{-0.5\lambda_1}}{1-\delta} + \ln\frac{\lambda_1}{\lambda_2-\lambda_1} + \ln\epsilon)/(\ln\delta - 0.5\lambda_1) - 1, 8\frac{N_h-1}{\lambda_1}\}\rceil N_h) = \Theta(N_h \ln\frac{1}{\epsilon})$. This shows that Algorithm 3 is highly efficient. To apply Algorithm 3, we need to infer model parameters $\lambda_1, \lambda_2, N_h$ and Q from data (we will infer them in Section 7).

Algorithm 3. Evaluating Long Term Profit Gains

Input: Model parameters $\lambda_1, \lambda_2, N_h, \mathcal{R}, \gamma$ and δ. Accuracy factor ϵ. Intrinsic quality Q.
Output: $\hat{G}$

```
if R(arg min_{i∈{1,...,m}} |i − Q|) < γ then
    Ĝ = gλ2/(1−δ)
else
    Ĝ ← gλ2/(1−δ).
    t̂ = ⌈max{(ln((1−δe^{−0.5λ1})/(1−δ)) + ln(λ1/(λ2−λ1)) + ln ε)/(ln δ − 0.5λ1) − 1, 8(Nh−1)/λ1}⌉
    for t = 0 to t̂ do
        for k = 0 to Nh − 1 do
            Ĝ ← Ĝ + (λ1 − λ2) g δ^t e^{−λ1 t} (λ1 t)^k / k!
        end for
    end for.
end if
```

4.4 Summary

We developed a baseline data analytical framework to characterize the ramp up time T_r and the expected long term profit gains G from data. Note that our framework so far assumes a perfect scenario that customers never commit errors in perceiving service providers' intrinsic quality, i.e., $\hat{Q} = Q$. However, customers may commit errors due to human factors like biases, preferences, etc. We next extend our data analytical framework to incorporate such human factors.

5 Human Factors

We now present a probabilistic model to capture human factors in rating such as biases and preferences. To incorporate them into our data analytical framework (stated in Algorithm 1), and we show that it is computationally difficult to evaluate $E[T_r]$ and G from any data set due to human factors. This computational challenge motivates us to design efficient randomized algorithms which have theoretical performance guarantees to approximate $E[T_r]$ and G in Section 6.

5.1 Model for Human Factors

Customers may have personal preferences in expressing feedback ratings due to various human factors, e.g., inherent biases. More precisely, a critical customer may assign lower ratings while a lenient customer may assign higher ratings.

To illustrate, let us focus on just one service provider which we denote by S. S provides "high quality" ("low quality") services but may receive low (high) rating. We use the following probabilistic model to capture the collective rating behavior under such personal preferences

$$\Pr[S \text{ receives a level } i \text{ rating}] = \eta_i, \text{ for all } i = 1, \ldots, m,$$

where η_i denotes the probability of receiving a level i rating and $\sum_{i=1}^{m} \eta_i = 1$. One can vary the mean of the m-tuple $(\eta_1, \ldots, \eta_m)$ to reflect different level of personal preferences. The higher (lower) the mean implies that customers are more likely to be lenient (critical) ones. We point out that when all customers are unbiased, then $(\eta_1, \ldots, \eta_m)$ reflects the intrinsic quality of a service provider. The impact of inherent biases is to shift $(\eta_1, \ldots, \eta_m)$ towards a higher or lower mean. One of our objectives is to examine the impact of such human factors on the efficiency of online reputation system. We next extend Definition 6 to incorporate such human factors.

Definition 7. *In the presence of human factors, we say a service provider is intrinsically reputable if and only if $\sum_{i=1}^{m} \eta_i \mathcal{R}(i) \geq \gamma$, otherwise we say a service provider is intrinsically average.*

To incorporate them into the baseline data analytical framework (stated in Algorithm 1), we first need to infer some extra parameters, i.e., $\eta_1, \ldots, \eta_m$, in step 1 of Algorithm 1. Then in step 2 of Algorithm 1 we need to evaluate ramp up time and long term profit gains in the presence of human factors. We outline this extended data analytical framework in Algorithm 4.

Algorithm 4. Improved Data Analytical Framework

1: **Parameter inference.** (1) Infer rating distribution $\eta_1, \ldots, \eta_m$ from data. (2) Infer model parameters $m, \gamma, \mathcal{R}, N_h, \lambda_1$ and λ_2 from data.
2: **Quantifying system efficiency.** Evaluating the ramp up time T_r and expected long term profit gains G in the presence of human factors, i.e., $\eta_1, \ldots, \eta_m$.
3: **Uncover system deficiencies.** Uncover deficiencies of online reputation systems via studying the ramp up time T_r and expected long term profit gains G.

We will present the details of step 1 and step 3 of the above framework in Section 7. In this section we focus on addressing step 2 of Algorithm 4.

5.2 Two Rating Levels

We first consider a special case of two rating levels, i.e., $m = 2$. This will illustrate the key idea of our derivation as well as its underlying computational complexity.

- *Ramp up time:* Note that the ramp up time T_r is a random variable due to dynamics in a reputation

updating process. The following lemma states the closed-form expected ramp up time $E[T_r]$.

Lemma 1. *Suppose the number of rating levels is two, i.e., $m = 2$. The expected ramp time can be expressed as*

$$E[T_r] = \sum_{t=0}^{\infty} t \sum_{(N_1(0),N_2(0),\ldots,N_1(t-1),N_2(t-1))\in\mathbb{N}^{2t-2}} \prod_{\ell=0}^{t-1} \phi(N_1(i), N_2(i), \lambda(i)) \mathbf{I}_{\{r(t)\geq\gamma, n_1(t)+n_2(t)\geq N_h\}} \left(1 - \mathbf{I}_{\{\exists j<t, r(t)\geq\gamma, n_1(j)+n_2(j)\geq N_h\}}\right), \tag{12}$$

where $\lambda(t)$ is derived in Equation (5), and function ϕ is $\phi(x, y, z) = \binom{x+y}{y}\eta_1^x\eta_2^y e^{-z} z^{x+y}/(x + y)!$.

Theorem 5. *The computational complexity of evaluating $E[T_r]$ derived in Equation (12) is $\Omega(\sum_{t=0}^{\infty} t)$.*

- *Long term profit gains:* Note that $G \triangleq E\left[\sum_{t=0}^{\infty} \delta^t g N(t)\right] = \sum_{t=0}^{\infty} \delta^t g E[N(t)]$. This implies that we have to compute $E[N(t)]$ for all $t = 0, 1, \ldots, \infty$. The following lemma states the analytical expression for G.

Lemma 2. *When the number of rating levels is two ($m = 2$), the long term profit gains G can be expressed as*

$$G = \sum_{t=0}^{\infty} \delta^t g \Bigg\{ \lambda_1 \Bigg\{ \sum_{n_1(t)=0}^{\infty} \sum_{n_2(t)=0}^{\lfloor n_1(t)\alpha \rfloor} \sum_{\sum_{i=0}^{t-1} N_1(i)=n_1(t)} \sum_{\sum_{i=0}^{t-1} N_2(i)=n_2(t)} + \sum_{n_1(t)+n_2(t)<N_h} \sum_{\sum_{i=0}^{t-1} N_1(i)=n_1(t)} \sum_{\sum_{i=0}^{t-1} N_2(i)=n_2(t)} - \sum_{n_1(t)=0}^{N_h} \sum_{n_2(t)=0}^{\min\{\lfloor n_1(t)\alpha \rfloor, N_h - n_1(t)\}} \sum_{\sum_{i=0}^{t-1} N_1(i)=n_1(t)} \sum_{\sum_{i=0}^{t-1} N_2(i)=n_2(t)} \Bigg\} + \lambda_2 \Bigg\{ \sum_{n_1(t)=0}^{\infty} \sum_{n_2(t)=\max\{N_h - n_1(t), \lceil n_1(t)\alpha \rceil\}}^{\infty} \sum_{\sum_{i=0}^{t-1} N_1(i)=n_1(t)} \sum_{\sum_{i=0}^{t-1} N_2(i)=n_2(t)} \Bigg\} \Bigg\} \prod_{i=0}^{t-1} \phi(N_1(i), N_2(i), \lambda(i)), \tag{13}$$

where $\lambda(t)$ is derived in Equation (5), function ϕ is $\phi(x, y, z) = \binom{x+y}{y}\eta_1^x\eta_2^y e^{-z} z^{x+y}/(x + y)!$, and $\alpha = \frac{\gamma - \mathcal{R}(1)}{\mathcal{R}(2) - \gamma}$.

Theorem 6. *The computational complexity of evaluating G derived in Equation (13) is $\Omega(\sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \binom{i+j}{j})$.*

- *Summary of observations:* Let us summarize our observations thus far in analyzing the special case of two rating levels ($m = 2$): (1) We derived analytical expressions for $E[T_r]$ and G; (2) the analytical expressions indicate extremely large computational complexity (based on Theorems 5 and 6), and they are computationally infeasible. To overcome such problem, we propose efficient randomized algorithms (in Section 6) which have theoretical performance guarantees in computing $E[T_r]$ and G.

5.3 Extensions to More Than Two Rating Levels

One can extend Lemmas 1 and 2 to obtain closed-form expressions for $E[T_r]$ and G. Due to page limit, we will not present the derivation here but it is reasonable to expect that the underlying complexity is huge, so it makes naive computation of $E[T_r]$ and G impractical. Let us focus on developing a practical approach to tackle the challenges in evaluating $E[T_r]$ and G.

6 Randomized Algorithms

We showed in the last section that it is computationally infeasible to evaluate $E[T_r]$ and G in the presence of human factors. Here, we propose computationally efficient randomized algorithms which have theoretical performance guarantees to approximate $E[T_r]$ and G.

6.1 Approximating Ramp Up Time

One can compute $E[T_r]$ via stochastic monte carlo methods [19]. The basic idea of stochastic monte carlo methods is estimating a probabilistic metric via sample average. Specifically, we can simulate the reputation updating process for $K \in \mathbb{N}$ rounds. Each round produces one sample of the ramp up time (T_r). We use the average of these K samples, which we denote as $\hat{E}[T_r]$, to estimate $E[T_r]$. For each sample path of T_r, we simulate the reputation updating process until a service provider ramps up. The stochastic monte carlo method is depicted in Algorithm 5. More concretely, line 2 initializes the setting to consider a new seller. Line 3 checks whether the ramp up condition is satisfied and it stops the iteration until the condition is satisfied. Line 6 generates the number of transactions and updates the total number of ratings. Line 7 to 10 generate a feedback rating for each transaction. Line 11 updates the average score.

Algorithm 5. Randomized Algorithm for $E[T_r]$

Input: Model parameters $\eta_1, \ldots, \eta_m$, λ_1, N_h, m and $\mathcal{R}$.
Output: $\hat{E}[T_r]$

```
1: for i = 1 to K do
2:   r ← 0, n_1 ← 0, ..., n_m ← 0, T_r^i ← 0
3:   while r < γ or Σ_{ℓ=1}^m n_ℓ < N_h do
4:     T_r^i ← T_r^i + 1
5:     λ ← λ_1
6:     N ~ Poisson(λ)
7:     for j = 1 to N do
8:       ℓ ~ Multinomial(η_1, ..., η_m)
9:       n_ℓ ← n_ℓ + 1
10:    end for
11:    r ← Σ_{k=1}^m n_k R(k) / Σ_{ℓ=1}^m n_ℓ
12:  end while
13: end for
14: Ê[T_r] ← Σ_{i=1}^K T_r^i / K
```

We next analyze the computational complexity (Theorem 7) of Algorithm 5 and derive the number of simulation rounds (K) needed to guarantee an accurate value of $E[T_r]$ (Theorem 8).

Theorem 7. *Suppose a service provider is intrinsically reputable, i.e., $\sum_{i=1}^m \eta_i \mathcal{R}(i) \geq \gamma$. The expected computational complexity for Algorithm 5 is $O(KN_h + K\frac{(\mathcal{R}(m) - \mathcal{R}(1))^4}{(\sum_i \eta_i \mathcal{R}(i) - \gamma)^4})$.*

Theorem 8. *Suppose a service provider is intrinsically reputable, i.e., $\sum_{\ell=1}^m \eta_\ell \mathcal{R}(\ell) > \gamma$. If the number of simulation rounds satisfies $K = O(\frac{1}{\xi\epsilon^2 N_h^2}(\frac{\mathcal{R}(m) - \mathcal{R}(1)}{\sum_{\ell=1} \eta_\ell \mathcal{R}(\ell) - \gamma})^6)$, then Algorithm 5 guarantees that $|\hat{E}[T_r] - E[T_r]| \leq \epsilon E[T_r]$ with probability of at least $1 - \xi$, where $\xi \geq 0$ denotes the fail probability.*

To illustrate the bound of K. Let us consider $\xi = 0.1$ and $\epsilon = 0.1$, i.e., to guarantee the approximation error is less

than $0.1E[T_r]$ with probability of at least 0.9. Suppose $m = 5$, $\mathcal{R}(i) = i$, $\sum_{\ell=1} \eta_\ell \mathcal{R}(\ell) - \gamma = 0.2$ and $N_h = 100$. Then we have that $K = 2.44 \times 10^7$.

Remark. Algorithm 5 is computationally efficient and can determine $E[T_r]$ with arbitrarily small error.

6.2 Approximating Long Term Profit Gains

We compute G via stochastic monte carlo methods. We simulate our model for $K \in \mathbb{N}$ rounds. Each round produces one sample of the long term profit gains G. The average of these K samples, which we denote as $\hat{G}$, to estimate G. Note that to obtain one sample of G, one needs to simulate all transactions completed in time slot $0, 1, \ldots, \infty$. This is computationally expensive (and theoretically infeasible). To address this challenge, we show that one only needs to simulate M time slots, i.e., time slot 0 to M, and can tightly "bound" the error in estimating G. This stochastic monte carlo method is presented in Algorithm 6. More concretely, line 2 initializes the setting to consider a new seller. Line 3 stops the loop when the number of time slots hits M. Line 4 to 7 generate the transaction rate. Line 8 generates the number of transactions in a time slot. Line 9 updates the long term profit gain. Line 10 to 13 generates the feedback rating for each transaction and updates the total number of ratings. Line 14 updates the average score.

Algorithm 6. Randomized Algorithm for G

Input: Model parameters $\eta_1, \ldots, \eta_m$, λ_1, N_h, m and $\mathcal{R}$.
Output: $\hat{G}$

1: **for** $i = 1$ to K **do**
2: $r \leftarrow 0, n_1 \leftarrow 0, \ldots, n_m \leftarrow 0, G_i \leftarrow 0$
3: **for** $t = 0$ to M **do**
4: **switch** $(r, \sum_{\ell=1}^{m} n_\ell)$ **do**
5: **case** $r < \gamma$ or $\sum_{\ell=1}^{m} n_\ell < N_h$: $\lambda = \lambda_1$
6: **case** $r \geq \gamma$ and $\sum_{\ell=1}^{m} n_\ell \geq N_h$: $\lambda = \lambda_2$
7: **ends switch**
8: $N \sim Poisson(\lambda)$
9: $G_i \leftarrow G_i + Ng\delta^t$
10: **for** $j = 1$ to N **do**
11: $\ell \sim Multinomial(\eta_1, \ldots, \eta_m)$
12: $n_\ell \leftarrow n_\ell + 1$
13: **end for**
14: $r \leftarrow \sum_{k=1}^{m} n_k \mathcal{R}(k) / \sum_{\ell=1}^{m} n_\ell$
15: **end for**
16: **end for**
17: $\hat{G} \leftarrow \sum_{i=1}^{K} G_i / K$

We now analyze the computational complexity (Theorem 9) of Algorithm 6 and derive the appropriate K and M to guarantee an accurate approximation of G (Theorem 10).

Theorem 9. *The expected computational complexity for Algorithm 6 is $O(KM\lambda_2)$.*

Theorem 10. *If the number of simulation rounds satisfies $K = O(\frac{1}{\epsilon^2}\frac{\lambda_2}{\lambda_1^2}\frac{1}{\xi})$, and M satisfies $M = O(\ln\frac{\epsilon\lambda_1}{\lambda_2} / \ln\delta)$, then Algorithm 6 guarantees that $|\hat{G} - G| \leq \epsilon G$ holds with probability of at least $1 - \xi$.*

Remark. Algorithm 6 is computationally efficient and can compute G with arbitrarily small error.

TABLE 2
Statistics for Reputation Rating Datasets

	# of service providers	# of ratings
eBay	4,586	19,217,083
Helpouts	858	10,454
Amazon	32,888	5,066,070
TripAdvisor	11,543	3,114,876

7 Experiments on Real-World Data

We present experimental results on reputation rating for the datasets from eBay and Google Helpouts. We first infer model parameters from the data, and input these inferred values to our framework so to characterize the ramp up time and long term profit gains. We show that the existing ramp up time in eBay and Google Helpouts are long: around 791 days and 1327 days respectively. This shows the inefficiency of online reputation systems since the ramp up time can significantly influence the long term profit gains, i.e., a 80 percent reduction on ramp up time leads to 80 percent (eBay) and 100 percent (Google Helpouts) improvement in long term profit gains respectively. Lastly, we discover from the data that around 78.7 percent sellers have ramped up in eBay, but only 1.5 percent workers have ramped up in Google Helpouts.

7.1 Datasets

We crawled historical online reputation data (refer to Table 2). from four representative applications of reputation systems, i.e., electronic commerce (eBay), crowdsourcing (Google Helpouts), product review (Amazon), travel website (TripAdvisor).

eBay. Eay is a popular electronic commerce system, where buyers purchase products from online stores and when a transaction is completed, a buyer expresses a rating to indicate whether a seller is trustworthy or not. It deploys a three-level cardinal metric, i.e., $\{-1$ ("negative"), 0 ("neutral"), 1 ("positive")$\}$. Ratings are public and accessible to all buyers and sellers. We crawled the historical ratings of 4,586 sellers received from the first day that a seller joins the eBay till April 2013.

Google Helpouts. Google Helpouts provides online crowdsourcing services. In Google Helpouts, service providers (or workers) offer various types of services, e.g., teaching piano, teaching cooking, etc. Workers advertise the service that they can provide, e.g., a service provider provides piano teaching service. Requesters select workers to provide a service based on workers' reputation. When a transaction is completed, requesters express a rating to indicate the quality of the service using a five-level cardinal rating metric {1 ("Terrible"), 2 ("Poor"), 3 ("Average"), 4 ("Very good"), 5 ("Excellent") }. Ratings and overall rating statistics for each worker are public and accessible to all users. We crawled historical transactions of 858 workers received from the first day that a worker joins Google Helpouts till January 2015.

TripAdvisor. TripAdvisor is a popular travel website, where users express ratings to hotels, restaurants, etc., to reflect the quality (or reputation) of these items. It uses the

TABLE 3
Inferred γ, N_h, λ_1 and λ_2

	m	γ	N_h	λ_1	λ_2
eBay	3	0.5	100	0.1742	2.4883
Helpouts	5	4	100	0.0689	0.4076
Amazon	5	4	100	0.1067	0.8916
TripAdvisor	5	4	100	0.0723	0.3831

TABLE 4
Inferred m and $\eta_1, \ldots, \eta_m$

	$\{\eta_1, \ldots, \eta_m\}$
eBay	$\{0.0023, 0.0034, 0.9943\}$
Helpouts	$\{0.0150, 0.0039, 0.0211, 0.0950, 0.8650\}$
Amazon	$\{0.0452, 0.0335, 0.0674, 0.1967, 0.6572\}$
TripAdvisor	$\{0.0172, 0.0295, 0.0822, 0.3242, 0.5468\}$

same rating system as that of Google Helpouts. We crawled ratings of 11,543 hotels received from the first day that a hotel joins this web site till April 2013.

Amazon. Amazon is a typical product review system, where users express ratings (or reviews) on products to reflect the product quality (or reputation). It uses the same rating system as that of Google Helpouts. We crawled ratings of 32,888 products received from the first day that a product joins this web site till April 2013.

7.2 Inferring Model Parameters

We now infer the parameters of our model, which are summarized in Tables 4 and 3. A time slot is one day. From Table 2 we observe that eBay adopts a three-level rating metric and the other three adopt a five-level rating metric respectively. Hence we have $m = 3$ for eBay and $m = 5$ for Google Helpouts, Amazon and TripAdvisor. Furthermore, from Table 2, we obtain the rating map $\mathcal{R}$ as $\{\mathcal{R}(1) = -1, \mathcal{R}(2) = 0, \mathcal{R}(3) = 1\}$ for eBay and $\{\mathcal{R}(1) = 1, \ldots, \mathcal{R}(5) = 5\}$ for Google Helpouts, Amazon and TripAdvisor. Each rating level is associated with a physical meaning, e.g., in eBay, we have {-1 ("negative"), 0 ("neutral"), 1 ("positive")}. We therefore set the reputation threshold as $\gamma = \frac{0+1}{2} = 0.5$. Similarly, for Google Helpouts, Amazon and TripAdvisor we have {1 ("Terrible"), 2 ("Poor"), 3 ("Average"), 4 ("Very good"), 5 ("Excellent")}. We set the reputation threshold as $\gamma = 4$. Xie and Lui studied rating sufficiency conditions for online rating systems [31], and they revealed that around 100 ratings can reflect the true quality of a product. In other words, the aggregate rating is statistically significant if an service provider has around 100 ratings. We therefor set $N_h = 100$. We will justify that this selection on γ is reasonable in Section 7.3 via extensive experiments on our datasets. Apply the inferred γ and N_h on our datasets, we infer the transactions' rate λ_1 (λ_2) as average number of transactions' per day completed by "average" ("reputable") service providers

$$\lambda_1 = \frac{\#[\text{transations by average service providers}]}{\#[\text{days to accumulate these transactions}]}, \tag{14}$$

$$\lambda_2 = \frac{\#[\text{transations by reputable service providers}]}{\#[\text{days to accumulate these transactions}]}. \tag{15}$$

Applying these two rules on our data set, we obtain the transactions' rate before ramping up and after ramping up. We summarize them in Table 3. Note that η_i denotes the probability of an intrinsically reputable service provider receives a level i rating. We say a service provider is intrinsically reputable if it has at least $N_h = 100$ ratings and its average rating is above the inferred γ. We therefore infer η_i as the fraction of level i ratings across all intrinsically reputable service providers

$$\eta_i = \frac{\#[\text{level } i \text{ ratings across all intrinsically reputable SPs}]}{\#[\text{ratings across all intrinsically reputable SPs}]},$$

where *SPs* denote service providers for short. Performing this rule on our datasets, we have η_i presented in Table 4.

7.3 Justifications of the Inferred Parameters

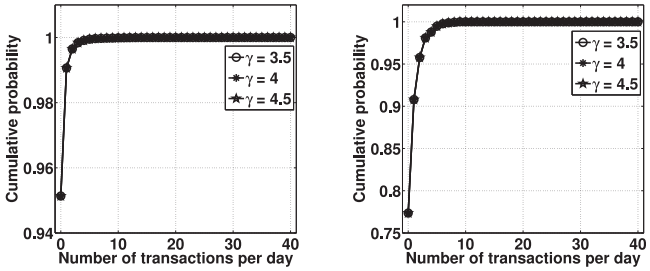
We conduct extensive experiments to justify that the inferred value of two building block parameters, i.e., γ and N_h are reasonable. These two parameters determine all other parameters like transactions' rate, the cumulative probability mass function of the number of transactions that arrive to a service provider per day. Our experimental results show that the inferred value of γ is an accurate estimation on its true value and the inferred threshold N_h is a typical value and selecting it does not loss any generality.

In particular, we study the impact of γ and N_h on the distribution of the the number of transactions per day to a service provider. This is because this distribution is the most fundamental one, since it determines the transactions rate and the per-day profit gains. With the above inferred parameters, we infer the cumulative probability mass function of the number of transactions that a service provider (average service provider and reputable service provider) receives in one day from data. Consider average labeled service providers, we infer the corresponding probability mass function via computing the fraction of days that they receive at most i transactions, where $i = 0, 1, \ldots, \infty$, i.e.,

$$\Pr[\text{ASP receive at most } i \text{ transactions}] = \frac{\#[\text{days that ASP receive at most } i \text{ transactions}]}{\sum_j \#[\text{days that ASP receive } j \text{ transactions}]},$$

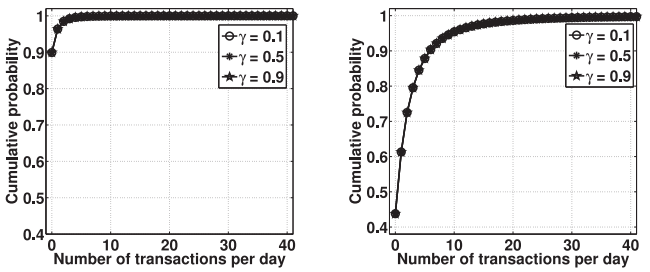
where ASP refers to average service providers. Similarly, we infer the cumulative probability function for reputable service providers.

We first study the impact of γ on the inferred the cumulative probability mass function of the number of transactions per day to a service provider almost remain unchanged. For Google Helpouts, Amazon and TripAdvisor we vary the value of γ from small to large, i.e., $\gamma = 3.5$, $\gamma = 4$ and $\gamma = 4.5$. Notice that in experience $\gamma = 3.5$ ($\gamma = 4.5$) is so small (large) for Google Helpouts that the true value of γ should be smaller (larger) than it. For eBay, we vary the value of γ from small to large, i.e., $\gamma = 0.1$, $\gamma = 0.5$ and $\gamma = 0.9$. Notice that in experience $\gamma = 0.1$ ($\gamma = 0.9$) is so small (large) for Google Helpouts that the true value of γ should be smaller (larger) than it. For each value of γ we infer the cumulative probability mass function, setting N_h



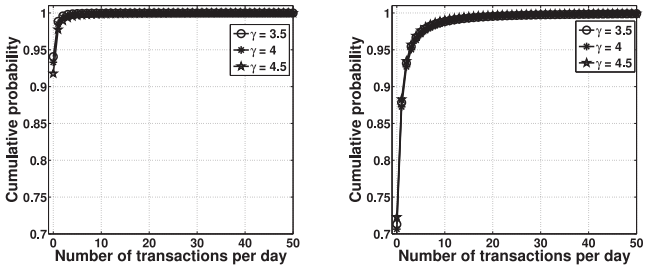
(a) Before Ramping Up (b) After Ramping Up

Fig. 1. Impact of γ on the transactions' distribution in Google Helpouts.



(a) Before Ramping Up (b) After Ramping Up

Fig. 2. Impact of γ on the transactions' distribution in eBay.

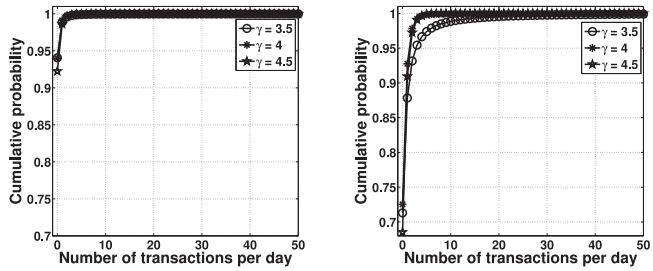


(a) Before Ramping Up (b) After Ramping Up

Fig. 3. Impact of γ on the transactions' distribution in Amazon.

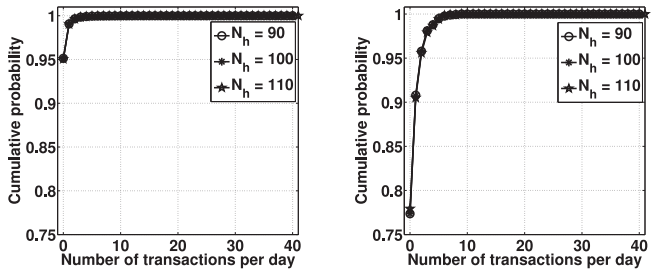
as that inferred in Section 7.2. Fig. 1 depicts the cumulative probability functions for Google Helpouts, where the horizontal axis presents the number of transactions that a service provider receives in one day, and the vertical axis shows the corresponding cumulative mass probability. Consider average labelled service providers in Google helpouts, Fig. 1a presents the probability mass functions for average labeled service providers in Google helpouts. It contains three curves corresponding to $\gamma = 3.5, \gamma = 4$ and $\gamma = 4.5$ respectively. One can observe that these three curves overlap. This means that the cumulative probability mass function for average labeled service providers in Google Helpouts remains unchanged as we vary γ from small to large. This statement also holds for reputable service providers in Google helpouts as one can observe in Fig. 1b. Therefore, for Google helpuots the cumulative probability function for the number of transactions that a service provider receives in one day remains unchanged as γ varies from small to large. This shows that the inferred γ is an accurate estimation on its true value. This statement also holds for the eBay, Amazon and TripAdvisor dataset, as shown in Figs. 2, 3, and 4.

We now study the impact of N_h on the inferred the cumulative probability mass function of the number of



(a) Before Ramping Up (b) After Ramping Up

Fig. 4. Impact of γ on the transactions' distribution in TripAdvisor.



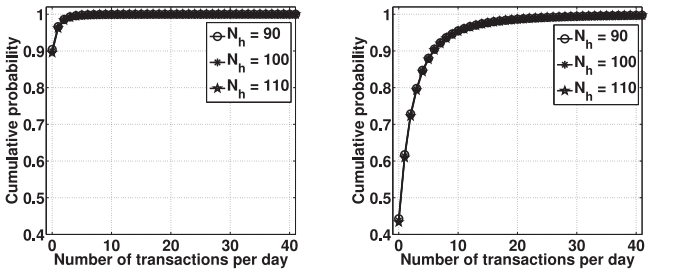
(a) Before Ramping Up (b) After Ramping Up

Fig. 5. Impact of N_h on the transactions' distribution in Google Helpouts.

transactions per day to a service provider. We will show that as we perturb N_h by one percent from its inferred value in Section 7.2, the inferred model parameters, i.e., transactions' rate λ_1, λ_2 and the cumulative probability mass function of the number of transactions per day, varies slightly. This means that the inferred N_h is a representative value and selecting it does not loss any generality. Again, it boils down to show that the cumulative probability mass function varies slightly. We perturb the value of N_h by one percent from its inferred value, i.e., set $N_h = 90, 100$ and 110. For each value of N_h we infer the cumulative probability mass function of the number of transactions received in one day setting γ as that inferred in Section 7.2. Fig. 5 depicts the cumulative probability functions inferred form the Google Helpouts dataset, where the horizontal axis presents the number of transactions that a service provider receives in one day, and the vertical axis shows the corresponding cumulative mass probability. Fig. 5a presents the corresponding probability mass functions for average labeled service providers. It contains three curves corresponding to $N_h = 90, N_h = 100$ and $N_h = 110$ respectively. One can observe that these three curves almost overlap. This means that the cumulative probability mass function for average labeled service providers in Google Helpouts varies slightly as we perturb N_h by one percent from its inferred value. This statement also holds for reputable service providers in Google helpouts as shown in Fig. 5b. Therefore, for Google helpuots the cumulative probability function for the number of transactions per day varies slightly as as we perturb N_h by one percent from its inferred value. The same observations can be obtained for the eBay, Amazon and TripAdvisor dataset as shown in Figs. 6, 7, and 8.

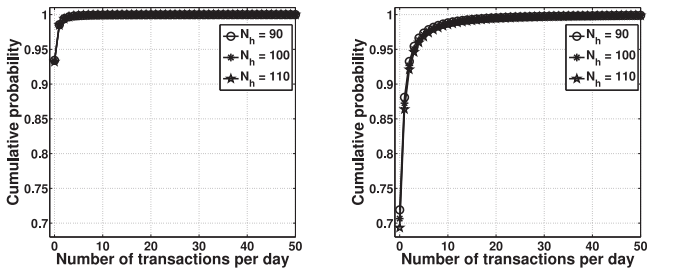
7.4 Model Validation

We validate that the number of transactions that a service provider receives in one day follows a Poisson distribution.



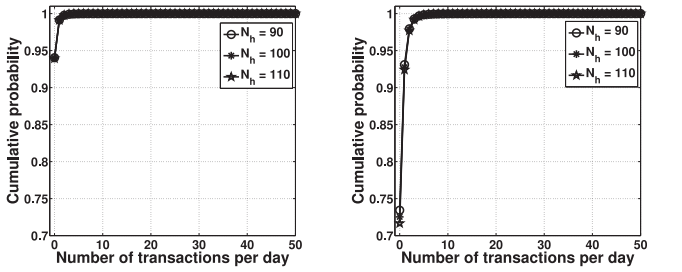
(a) Before Ramping Up (b) After Ramping Up

Fig. 6. Impact of N_h on the transactions' distribution in eBay.



(a) Before Ramping Up (b) After Ramping Up

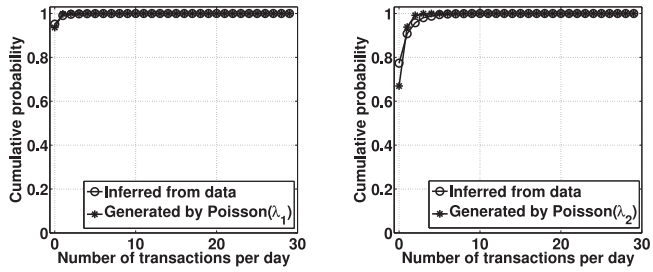
Fig. 7. Impact of N_h on the transactions' distribution in Amazon.



(a) Before Ramping Up (b) After Ramping Up

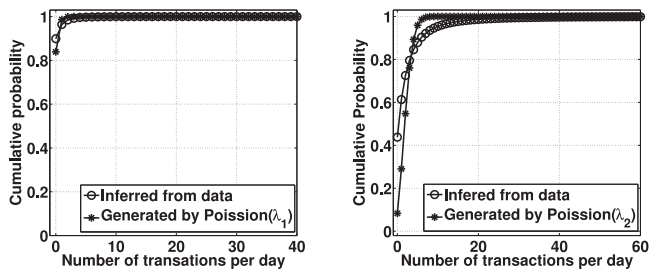
Fig. 8. Impact of N_h on the transactions' distribution in TripAdvisor.

In particular, we show that the inferred probability mass function are almost the same as the probability mass function of a Poisson distributions with the inferred transaction rates λ_1 and λ_2. Fig. 9 depicts the cumulative probability mass functions, where the horizon axis presents the number of transactions that a service provider receives in one day and the vertical axis shows the corresponding cumulative probability. It has two sub-figures corresponding to average labeled and reputable service provider respectively. Each sub-figure contains two curves corresponding cumulative probability mass functions inferred from day and generated by the Poisson distribution respectively. From Fig. 9a we observe that these two curves almost overlap. This shows that for average labeled service providers in Google helpouts, the distribution of the number of transactions per day follows a Poisson distribution. This statement also holds for reputable service providers as one can observe from Fig. 9b. Therefore for Google Helpouts, the distribution of the number of transactions per day follows a Poisson distribution. This statement also holds for the eBay, Amazon and TripAdvisor dataset as shown in Figs. 10, 11, and 12.



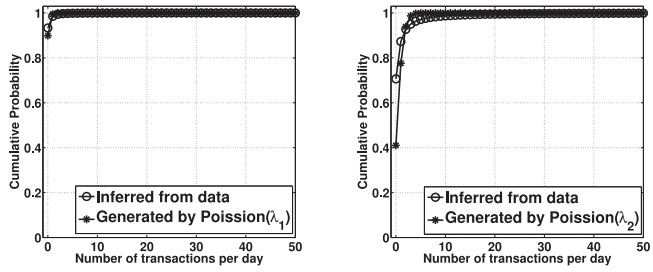
(a) Before Ramping Up (b) After Ramping Up

Fig. 9. Transaction distribution validation for Google Helpouts.



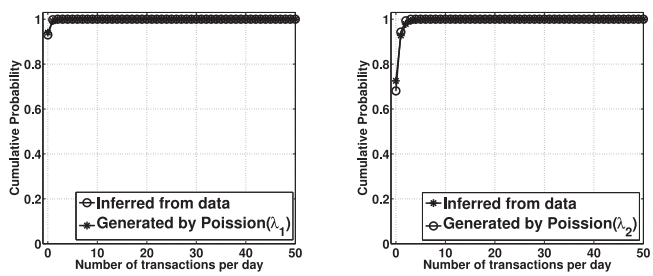
(a) Before Ramping Up (b) After Ramping Up

Fig. 10. Transaction distribution validation for eBay.



(a) Before Ramping Up (b) After Ramping Up

Fig. 11. Transaction distribution validation for Amazon.



(a) Before Ramping Up (b) After Ramping Up

Fig. 12. Transaction distribution validation for TripAdvisor.

7.5 Characterizing Ramp Up Time

We apply Algorithm 5 to compute the ramp up time. Applying Theorems 7 and 8, we set $K = 10^8$ since it guarantees $|\hat{E}[T_r] - E[T_r]| \leq 0.01E[T_r]$ with probability at least 0.99. We input the above inferred parameters to Algorithm 5 and obtain the expected ramp up times for eBay, Google Helpouts, Amazon and TripAdvisor respectively and they are stated in Table 5. From Table 5 we observe that for Google Helpouts the expectation of the ramp up time is 1454 days. This means that on average, a worker needs to

TABLE 5
Expected Ramp Up Time $E[T_r]$

	Helpouts	eBay	Amazon	TripAdvisor
$E[T_r]$	1454 (days)	576 (days)	943 (days)	1383 (days)

spend 1454 days to get ramped up. This is quite a long time, which may lead to that some workers get dropped out and discourages new workers to join. The implication for Google Helpouts is that the reputation imposes a negative impact on increasing the user population and it needs extra strategies to recruit new workers effectively so as to increase user population in its early stage. Consider eBay, the expectation of the ramp up time is $E[T_r] = 576\,(\text{days})$. Namely, on average, it takes 576 days for a seller to get ramped up, which is a long duration. As a consequence, some sellers may get dropped out before getting ramped up or they shit to other online selling platforms and new sellers may get discouraged to join. The implication for the eBay is that it needs to deploy some mechanisms to help new sellers to attract buyers so as to get ramped up quickly. For Amazon and TripAdvisor, the expected ramp time are 943 days and 1383 days respectively. This implies that the Amazon (or TripAdvisor) website needs to deploy incentive mechanisms to attract users to assign ratings or reviews to new products (or hotels) so that high quality products can be identified in a shorter time. This will benefit users which in turn benefit the website by attracting more users. Lastly, from Table 5, we also observe that the ramp up time for eBay is the shortest and the ramp up time for Google Helpouts is the longest. This implies that the online reputation system in eBay is the most efficient one while the online reputation system in Google Helpouts is the most inefficient one.

We now apply our framework to investigate the fraction of service providers that have got ramped up. This fraction service providers is an important indicator for an Internet service platform and based on it we obtain deeper insights and implications to improve the Internet service platforms. Formally, let f_{ramp} denote fraction of service providers that have got ramped up

$$f_{ramp} \triangleq \frac{\#[\text{service providers having } r \geq \gamma, \sum_{i=1}^{m} n_i \geq N_h]}{\#[\text{service providers}]}.$$

Recall that the inferred condition for a service provider to get ramped up is $(\gamma, N_h) = (0.5, 100)$ for eBay, and $(\gamma, N_h) = (4, 90)$ for Google Helpouts, Amazon and TripAdvisor. Applying this condition to our dataset, we obtain the empirical values of f_{ramp}, which is presented in Table 6. From Table 6 we observe that for Google Helpouts we have $f_{ramp} = 1.5\%$. Namely, only 1.5 percent of workers have got ramped up. One possible reason is that Google Helpouts is only around one year old and its ecosystem is still at the infancy. Recall that the ramp up time is quite long, i.e., 1454 days. The long ramp up time and small f_{ramp} uncovers

TABLE 6
Fraction of Ramped Up Service Providers

	Helpouts	eBay	Amazon	TripAdvisor
f_{ramp}	1.4 %	84.5 %	21.0%	26.8%

TABLE 7
Long Term Profit Gains ($g=1, \delta=0.999$)

	G	$G_{\max}$	$G/G_{\max}$
Helpouts	147.5	402.8	36.61%
eBay	1477.3	2472.6	59.74%
Amazon	413.8	884.0	46.81%
TripAdvisor	151.1	377.6	40.01%

uncovers a deficiency of the reputation system of Google Helpouts in increasing the number of workers. This deficiency uncovers a key reason why Google Helpouts was eventually shut down in April 2015. Consider eBay, we observe that $f_{ramp} = 84.5\%$. This means that a large fraction of users have got ramped up. One reason is that eBay is over ten years old and sellers who remain in eBay have sufficient time to ramp up. There are still many workers have not got ramped up, i.e, around 16 percent. Recall that ramp up time is also long, i.e., 576 days. This uncovers a key reason why eBay is under a significant user loss [12], [23], [26]. For Amazon and TripAdvisor we have $f_{ramp} = 21.0\%$ and $f_{ramp} = 26.8\%$. Namely, most products (or hotels) have not got ramped up yet in Amazon (or TripAdvisor). Recall that the ramp up time for Amazon and TripAdvisor are long, i.e., 943 and 1383 days respectively. This implies that to identify more high quality products (or hotels) in a shorter time, the Amazon (or TripAdvisor) website needs to incentivize users to assign ratings to averaged labeled products (or hotels).

7.6 Characterizing Long Term Profit Gains

Now we study the impact of ramp up time on the long term profit gains. In particular, we would like to know to what extend reducing ramp up time can improve the long term profit gains. Through this we reveal whether reducing ramp up time is meaningful for service providers. We apply Algorithm 6 to compute long term profit gains. We set the discounting factor as $\delta = 0.999$, the unit profit gain to be $g = 1$. Applying Theorems 9 and 10 we set $K = 10^8$ and $M = 50000$ since they guarantee $|\hat{G} - G| \leq 0.01G$ with probability at least 0.99. We input the inferred parameter into Algorithm 6 to compute G. To show the potential improvement of long term profit gains via reducing the ramp up time, we also compute the theoretical maximum long term profit gains denoted by $G_{\max}$, which is attained when N_h is equal to zero. Table 7 presents numerical results on $G, G_{\max}$ and $G/G_{\max}$. One can observe that for the long term profit gains, Google Helpouts only achieves 36.61 percent, eBay only achieves 59.74 percent, Amazon achieves 46.81 percent and TripAdvisor achieves 40.01 percent of its maximum possible value $G_{\max}$. This shows that there is a great opportunity to improve the long term profit gains via reducing the ramp up time. Namely, it meaningful to reduce the ramp up time for service providers.

Let us now study how the ramp up time influences the long term profit gains, since this will give us important insights on why some service providers drop out, and why some new service providers participate. We examine the impact of T_r on G by varying T_r. We consider the scenario that the Internet service company can control N_h, and we want to find our how G can be improved if we reduce T_r (T_r can be reduced by reducing N_h). We define reduction ratio

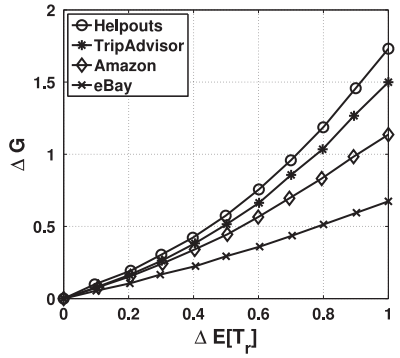


Fig. 13. Impact of ramp up time T_r on G.

of ramp up time

$$\Delta E[T_r] = (E[T_r] - \widetilde{E}[T_r])/E[T_r],$$

where $E[T_r]$ is the ramp up time without reduction on N_h, $\widetilde{E}[T_r]$ is the new ramp up time with some reduction on N_h. We define improvement ratio on the long term profit gains

$$\Delta G = (\widetilde{G} - G)/G,$$

where G is the long term profit gains without reduction on N_h, $\widetilde{G}$ is the new long term profit gains with some reduction on N_h. Fig. 13 shows the impact of $\Delta E[T_r]$ on ΔG, where the horizontal axis represent $\Delta E[T_r]$ and the vertical axis shows the corresponding ΔG. One can observe that as we reduce ramp up time, we increase the long term profit gains. When we reduce $E[T_r]$ by 80 percent, the improvement of G is around 100 percent for Google Helpouts, 50 percent for eBay, and around 80 percent for Amazon and TripAdvisor. It shows that reducing $E[T_r]$, we can significantly improve the long term profit gains.

7.7 Implications

Our findings can benefit both the practice and theory of reputation systems. For eBay like e-commerce systems they need to deploy some extra mechanisms or refine the design of their reputation system to reduce the ramp up time so that they can reduce the probability that sellers drop out and attract more sellers. For newly start up systems like Google helpouts, they need to to deploy some reputation systems having short ramp up time, because in the early stage ramp up users is critical. For practical reputation system design, they should be aware of the ramp up time. Lastly, our frameworks can serve as important building blocks to identify deficiencies of reputation systems.

8 Related Works

Reputation systems [21] is an important research topic in network economics. Research on reputation system can be categorized into three typical aspects: (1) reputation formulation and calculation, e.g., [11], [16], (2) attacks and defense techniques design e.g., [3], [10], and (3) effectiveness and efficiency of reputation systems [5]. A survey is given in [13].

Many theoretical works explored reputation metric formulation and calculation. There are two typical reputation formulating models, i.e., the rating-based model [11], [24] and the transitive trust model [3], [4]. The rating-based reputation formulating model aims to solicit explicit human feedbacks (or ratings) [2], [11], [24], [33]. It computes a reputation score for each user by summarizing his feedback ratings. The transitive trust reputation model [3], [4], [16], [22], [32] captures the propagation of trust among users. Graph model is applied to quantify users' reputation, i.e., each user is abstracted as a node, and each weighted directed link, e.g., from B to A, measures the trust that B expresses to A. Several algorithms were proposed to compute the reputation score for users [3], [4], [16], [22], [32]. The key difference between our work and theirs is that we conduct a data-oriented study to uncover the inefficiency of real-world online reputation systems, while theirs are theoretical in nature. Our work enriches theoretical studies by uncovering the importance of ramp up time.

A number of defense techniques have been developed for reputation systems. One typical potential attack is dishonest feedbacks. Peer-prediction mechanisms were proposed to address this attack [14], [15], [18]. Another potential attack is reputation inflation, and number of techniques have been proposed to address this attack [3], [10], [28], [32]. A nice survey on the state-of-the-art attack and defense techniques is [10]. Our work propose a general framework to investigate the efficiency of defense techniques. We propose an important factor, i.e., ramp up time, that various defense techniques need to be aware of.

Several works explored the effectiveness of reputation systems [1], [5], [6], [17]. In [5], authors tried to improve the efficiency of eBay reputation computation by proposing an algorithm which relies on buyer friendship to filter out unfair ratings. The work [5] explored how buyers' rating biases (i.e., leniency or criticality) may influence sellers' product advertising behavior in eBay. Authors in [17] conducted a measurement study on the impact of negative feedbacks on eBay reputation system. Our work is different form theirs in that their works only applies to eBay reputation system, while our work applies to general rating-based reputation systems. We examine ramp up time and propose randomized algorithms to carry out large scale data analytics to uncover the deficiencies of online reputation systems.

9 Conclusions

This is the first paper which presents a data driven approach to uncover the deficiencies of real-world online reputation systems. We proposed two measures to quantify the efficiency of online reputation systems: (1) ramp up time of a new service provider, (2) long term profit gains for service providers. We present a novel data analytical framework to evaluate these two measures from data. We showed that it is computationally infeasible to evaluate these two measures due to inherent preference or personal biases in expressing feedbacks (or ratings). We developed computationally efficient randomized algorithms with theoretical performance guarantees to address this computational challenge. We apply our methodology to real-life datasets from eBay, Google Helpouts, Amazon and TripAdvisor. We extensively validate our model. We discover the deficiencies of online reputation systems: (1) the ramp up time is more than 500 days; (2) reducing ramp up time can improve the long term profit gains significantly, e.g., an 80 percent

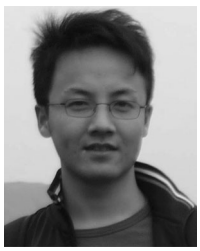
reduction on ramp up time leads to at least 50 percent (as high as 100 percent) improvement in long term profit gains. Our experimental results also uncover insights on why Google Helpouts was eventually shut down and why eBay is losing sellers heavily.

Acknowledgments

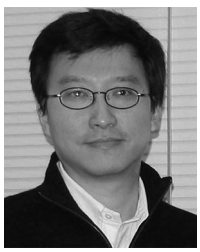
This work is supported in part by the GRF 14205114. An earlier conference version of the paper appeared in the IEEE International Conference on Data Mining (ICDM 2015) [30]. In this journal version, we add a novel measure, i.e., long term profit gains to quantify the efficiency of a reputation system. We propose a computationally efficient randomized algorithms with theoretical performance guarantees to approximate the long term profit gains. We conduct more experiments to validate our model and uncover the deficiencies of online reputation systems.

References

[1] R. Bhattacharjee and A. Goel, "Avoiding ballot stuffing in eBay-like reputation systems," in *Proc. ACM SIGCOMM Workshop Econ. Peer-to-Peer Syst.*, 2005, pp. 133–137.
[2] S. Buchegger and J.-Y. Le Boudec, "A robust reputation system for peer-to-peer and mobile ad-hoc networks," in *Proc. 2nd Workshop Econ. Peer-to-Peer Syst.*, 2004, pp. 120–125.
[3] A. Cheng and E. Friedman, "Sybilproof reputation mechanisms," in *Proc. ACM SIGCOMM Workshop Econ. Peer-to-Peer Syst.*, 2005, pp. 128–132.
[4] R. Delaviz, N. Andrade, J. Pouwelse, and D. Epema, "SybilRes: A sybil-resilient flow-based decentralized reputation mechanism," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 203–213.
[5] C. Dellarocas, "Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior," in *Proc. 2nd ACM Conf. Electron. Commerce*, 2000, pp. 150–157.
[6] C. Dellarocas, "Analyzing the economic efficiency of eBay-like online reputation reporting mechanisms," in *Proc. 3rd ACM Conf. Electron. Commerce*, 2001, pp. 171–179.
[7] eBay, 1995. [Online]. Available: http://www.ebay.com/
[8] Forbes, "Google helpouts is shutting down on april 20th," 2015. [Online]. Available: http://www.forbes.com/sites/amitchowdhry/2015/02/16/google-helpouts-is-shutting-down-on-april-20th/
[9] Google Helpouts, 2013. [Online]. Available: https://helpouts.google.com/
[10] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Comput. Surveys*, vol. 42, no. 1, pp. 1:1–1:31, 2009.
[11] D. Houser and J. Wooders, "Reputation in auctions: Theory, and evidence from eBay," *J. Econ. Manage. Strategy*, vol. 15, no. 2, pp. 353–369, 2006.
[12] IWantCollectibles, "Why most eBay sellers fail?" 2011. [Online]. Available: http://www.news.iwantcollectibles.com/ebay-sellers-fail.shtml
[13] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Syst.*, vol. 43, no. 2, pp. 618–644, 2007.
[14] R. Jurca and B. Faltings, "Minimum payments that reward honest reputation feedback," in *Proc. 7th ACM Conf. Electron. Commerce*, 2006, pp. 190–199.
[15] R. Jurca and B. Faltings, "Collusion-resistant, incentive-compatible feedback payments," in *Proc. 8th ACM Conf. Electron. Commerce*, 2007, pp. 200–209.
[16] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proc. 12th Int. Conf. World Wide Web*, 2003, pp. 640–651.
[17] T. Khopkar, X. Li, and P. Resnick, "Self-selection, slipping, salvaging, slacking, and stoning: The impacts of negative feedback at eBay," in *Proc. 6th ACM Conf. Electron. Commerce*, 2005, pp. 223–231.
[18] N. Miller, P. Resnick, and R. Zeckhauser, "Eliciting informative feedback: The peer-prediction method," *Manage. Sci.*, vol. 51, no. 9, pp. 1359–1373, 2005.
[19] M. Mitzenmacher and E. Upfal, *Probability and Computing*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[20] oDesk, 2009. [Online]. Available: https://www.odesk.com/o/home
[21] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, "Reputation systems," *Commun. ACM*, vol. 43, no. 12, pp. 45–48, 2000.
[22] P. Resnick and R. Sami, "Sybilproof transitive trust protocols," in *Proc. 10th ACM Conf. Electron. Commerce*, 2009, pp. 345–354.
[23] Retailwire, "Ebay's loss is amazon marketplace gain," 2015. [Online]. Available: http://www.retailwire.com/discussion/18203/ebays-loss-is-amazon-marketplaces-gain
[24] A. Singh and L. Liu, "TrustMe: Anonymous management of trust relationships in decentralized P2P systems," in *Proc. 3rd Int. Conf. Peer-to-Peer Comput.*, 2003, pp. 142–149.
[25] Taobao, 2003. [Online]. Available: http://www.taobao.com/
[26] Topix, "Ebay exodus: Why are thousands of sellers leaving eBay?" 2013. [Online]. Available: http://www.topix.com/forum/indy/TFE181M8MUGHKR73C
[27] TripAdvisor, 2000. [Online]. Available: http://www.tripadvisor.com/
[28] B. Viswanath, M. Mondal, K. P. Gummadi, A. Mislove, and A. Post, "Canal: Scaling social network-based Sybil tolerance schemes," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 309–322.
[29] WebProNews, "Top 10 frustrations for eBay sellers," 2009. [Online]. Available: http://www.webpronews.com/top-10-frustrations-for-ebay-sellers-2009–01
[30] H. Xie and J. C. S. Lui, "Data driven approach to uncover deficiencies in online reputation systems," in *Proc. IEEE Int. Conf. Data Mining*, 2015, pp. 1045–1050.
[31] H. Xie and J. C. S. Lui, "Mathematical modeling and analysis of product rating with partial information," *ACM Trans. Knowl. Discovery Data*, vol. 9, no. 4, pp. 26:1–26:33, 2015.
[32] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: Defending against Sybil attacks via social networks," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2006, pp. 267–278.
[33] R. Zhou and K. Hwang, "PowerTrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 460–473, Apr. 2007.



Hong Xie received the BEng degree from the School of Computer Science and Technology, The University of Science and Technology of China, in 2010 and the PhD degree from the Department of Computer Science & Engineering, The Chinese University of Hong Kong, in 2015, proudly under the supervision of Professor John C. S. Lui. He is currently a postdoctoral research fellow in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include data science, approximation algorithms, network economics, etc. He is a member of the IEEE.



John C. S. Lui received the PhD degree in computer science from UCLA. He is currently the Choh-Ming Li chair professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His current research interests include communication networks, system security (e.g., cloud security, mobile security, etc.), network economics, network sciences, large-scale distributed systems, and performance evaluation theory. He serves in the editorial board of the *IEEE/ACM Transactions on Networking*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *Journal of Performance Evaluation*, and the *International Journal of Network Security*. He was the chairman of the CSE Department from 2005-2011. He received various departmental teaching awards and the CUHK Vice-Chancellors Exemplary Teaching Award. He is also a co-recipient of the IFIP WG 7.3 Performance 2005 and IEEE/IFIP NOMS 2006 Best Student Paper Awards. He is an elected member of the IFIP WG 7.3, fellow of the ACM, the IEEE, and Croucher senior research fellow. His personal interests include films and general reading.

Model-as-You-Go for Choreographies: Rewinding and Repeating Scientific Choreographies

Andreas Weiß, Vasilios Andrikopoulos, Michael Hahn, and Dimka Karastoyanova

Abstract—Scientists are increasingly using the workflow technology as a means for modeling and execution of scientific experiments. Despite being a very powerful paradigm workflows still lack support for trial-and-error modeling, as well as flexibility mechanisms that enable the ad hoc repetition of experiment logic to enable, for example, the convergence of results or to handle errors. In this respect, in our work on enabling multi-scale/field (multi-*) experiments using choreographies of scientific workflows, we contribute a method comprising all necessary steps to conduct the repetition of choreography logic across all workflow instances participating in a multi-* experiment. To realize the method, we contribute i) a formal model representing choreography models and instances, including the re-execute and iterate operations for choreographies, and based on it ii) algorithms for determining the rewinding points, i.e., the activity instances where the rewinding has to stop and iii) enable the actual rewinding to a previous execution state and repetition of the choreography. We present the implementation of our approach in a message-based, service-oriented system that allows scientists to model, control, and execute scientific choreographies as well as perform the rewinding and repeating of choreography logic. We also provide an evaluation of the performance of our approach.

Index Terms—Ad Hoc changes, flexible choreography, workflow, multi-* experiment, choreography rewinding, choreography re-execution and iteration

1 Introduction

Workflow technology offers an approach for the design and implementation of *in-silico experiments* such as scientific simulations. By means of *scientific workflows*, it supports the goal of eScience to provide generic approaches and tools for scientific exploration and discovery in different fields of natural and social sciences [1]. More specifically, scientific workflows are used to specify the control and data flow of in-silico experiments and orchestrate scientific software modules and services. Through the use of workflow technology in eScience, a significant body of knowledge and tools from the business process management domain becomes available to natural scientists. However, at the same time scientists have different requirements on workflow modeling and enactment than users in the business domain. For instance, eScience experiments often demand a trial-and-error based modeling [2] supporting the use of incomplete, partially defined workflow models, or the repetition of the execution of specific experiment steps with different sets of parameters. In this context, natural scientists are both the designers and users of a workflow model.

In order to address these requirements, previous work [3] proposed the *Model-as-you-go* approach for workflows. A key aspect of this approach is that it hides the differences between workflow model and instance by abstracting from technical details such as deployment. This works towards creating the impression of one coherent experimentation phase where scientists can iteratively model and execute a scientific workflow. Model-as-you-go also supports two types of user-initiated operations allowing the *ad hoc repetition* of parts of the workflow model without the a priori definition of execution control artifacts [4]. The *iterate* operation allows repeating workflow logic without undoing previously completed work. This is helpful for scientists to enforce the convergence of results by repeating some steps of the scientific workflow. The *re-execute* operation allows repeating parts of already executed workflow logic after undoing, or compensating already completed work. This allows scientists, for example, to reset the execution environment in case of detected errors or even when a complete redo of a simulation is needed.

A known limitation of this approach, however, is the insufficient support for multi-scale/multi-field, also known as *multi-** experiments, that recent works are addressing [5], [6]. Multi-scale experiments couple different length or time scales within the same experiment, e.g., part of the overall simulation simulates a natural phenomenon on the atomic scale with nanometer length and transfers the results to another simulation application simulating structures with

- *A. Weiß and M. Hahn are with the Institute of Architecture of Applications Systems (IAAS), University of Stuttgart, Universitaetsstr. 38, Stuttgart 70569, Germany. E-mail: {andreas.weiss, michael.hahn}@iaas.uni-stuttgart.de.*
- *V. Andrikopoulos is with the University of Groningen, Nijenborgh 9, Groningen 9747AG, The Netherlands. E-mail: v.andrikopoulos@rug.nl.*
- *D. Karastoyanova is with the Kühne Logistics University (KLU), Großer Grasbrook 17, Hamburg 20457, Germany. E-mail: dimka.karastoyanova@the-klu.org.*

Manuscript received 8 Dec. 2016; revised 21 May 2017; accepted 19 July 2017. Date of publication 31 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Andreas Weiß.) Digital Object Identifier no. 10.1109/TSC.2017.2732988

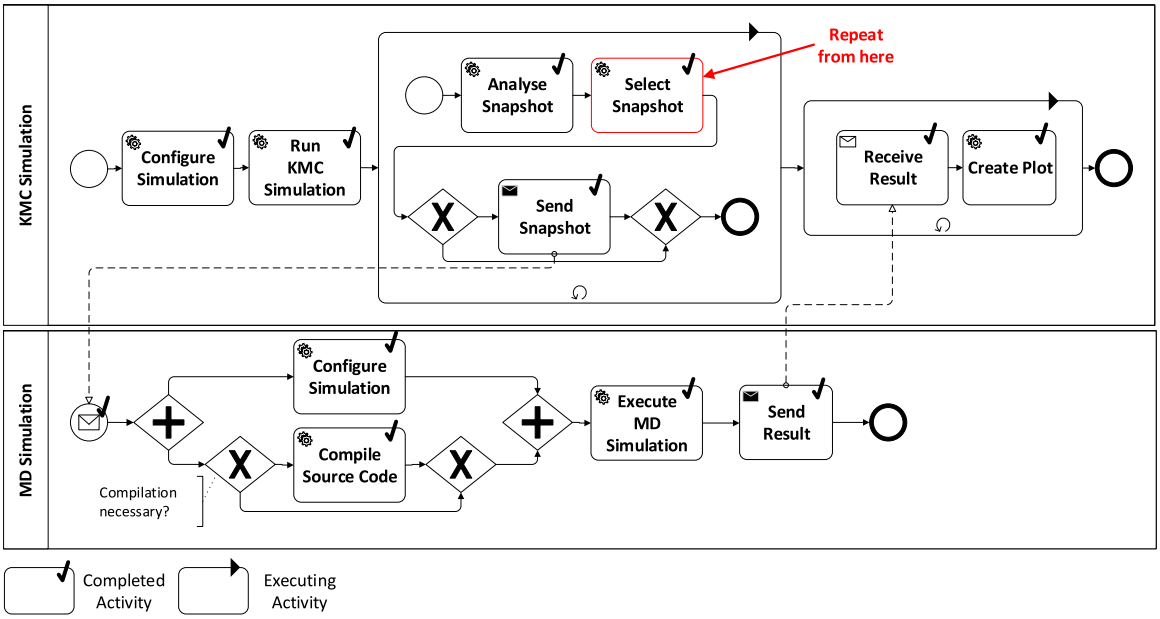


Fig. 1. An example of a choreographed multi-scale simulation of thermal aging of iron-copper alloys and their material behavior. Adapted from [5].

millimeter length. Multi-field experiments use different scientific fields in the same experiment, for example physics, biology, or chemistry. In order to provide better support for these types of in silico experiments, we proposed the use of choreographies [7]. Every scale and every distinct field is modeled as an independent choreography participant. This corresponds to the fact that multi-* experiments typically involve scientists from different disciplines and organizations having diverse expertise [5].

Fig. 1 shows a simplified example of a multi-scale experiment studying the simulation of thermal aging of iron-copper alloys and emerging effects on the mechanical behavior of the alloys [8]. The coupling of two simulation methods allows for carrying out simulations on multiple time and length scales. The first method is a Kinetic Monte Carlo (KMC) Simulation and simulates the formation of copper atom clusters (precipitates) in an atom lattice and stores the intermediate results in snapshots at discrete time steps. The generated snapshots are analyzed and sent to a Molecular Dynamics (MD) simulation workflow if the atom clusters have an appropriate size. The MD simulation workflow and the services implementing the activities apply forces on each snapshot to test material behavior after thermal aging. Each received snapshot triggers the creation of a new workflow instance of the MD simulation. The results are sent back to the KMC simulation workflow to generate an overall graphical plot for each simulation snapshot. Together, the two simulation workflows form a choreography with each simulation method being represented by an independent choreography participant.

Let's assume now that a scientist started the coupled simulation workflows and they already have calculated a simulation snapshot, applied forces on the atom lattice, and visualized the result graphically. The scientist discovers that the visualization does not show a plausible graph and wishes to re-run parts of the overall multi-scale simulation. These could be, for example, a change of the criteria that are used to select an appropriate KMC simulation snapshot, the re-sending of snapshots, and the re-run of the MD simulation. Essentially, this requires the capability that allows scientists to select a point in the KMC or MD workflows up to which the execution of the simulations has to be *rewound* before applying any desired changes and repeating the execution of this part of the simulation. Repeating part of the execution instead of discarding all intermediate results and starting from scratch saves a lot of time and effort for the scientists, especially in the case of typically long running experiments.

Toward supporting such type of control over scientific experiments, in this work we build on the concepts we introduced in previous work [4], [5], [6], and present the complete Model-as-you-go for Choreographies approach that allows for rewinding and repetition in choreographies. For this purpose, in Section 2 we present a *formal description of choreography models and instances*, which we extended from [6] to incorporate *loop constructs* typically used in acyclic choreography and workflow models. Furthermore, the formal model accounts for the so-called *participant sets* that can be used to model a participant that can be instantiated an arbitrary number of times. Subsequently, in Section 3, we introduce a new *method* comprising all steps necessary to rewind and repeat choreography logic, from determining the rewinding points, through rewinding the choreography, enacting a repeat or iteration of its logic, and to resuming the choreography execution. In the same section we present the *algorithm to determine the rewinding points*, which is a combination of its basic version as introduced in [6] and a significant extension to support loop activity instances and instances of participant sets. Rewinding points are activity instances in the participating workflows up to which the state has to be rewound and where the repetition begins. We enable the repetition or iteration of choreography logic by defining and implementing two operations: *iterate* and *re-execute*. For the actual rewinding at the choreography

participants, i.e., the resetting or compensation of activities in each involved participant instance, we resort to the concepts of [4], about which we provide the necessary background details. As with the previous works, the concepts introduced in this article are independent of a particular choreography or workflow language and therefore reusable across technologies. Section 4 describes how the concepts and algorithms for repetition are realized into a message-based and service-oriented system, the *ChorSystem*, that supports the modeling, execution, and control of scientific choreographies and implements the method we introduced in Section 3. In Section 5, the proposed approach is experimentally evaluated in terms of performance. Finally, Section 6 compares our approach to related ones and Section 7 concludes the article.

2 Formal Model

In this section, we define the underlying formal model for our approach in two parts: *modeling* and *execution*.

2.1 Modeling Phase

Typically, choreography models show only the publicly visible communication behavior, because the details of the workflows implementing the choreography participants are considered as sensitive information. The usually non-executable models are defined collaboratively and used to generate representations of the choreography participants in an executable workflow language. The collaborating organizations then refine the resulting workflow models they own with business logic [9].

A choreography model consists of at least two participants, which are represented by service orchestrations, workflow models, or process models. A process model is a directed, acyclic graph (DAG) whose nodes represent activities. Control flow is explicitly modeled by edges denoted as control flow connectors linking activities. Data flow is implicitly described through the manipulation of variables as input and output of activities. The participants communicate with each other via message links, a second type of edges.

A process model is formally defined as in [10]:

Definition 1 (Process Model, G). *A process model is a DAG $G = (m, V, i, o, A, L)$, where $m \in M$ is the name of the process model, $V \subseteq M \times S$ (M = set of names; S = set of data structures) is the set of variables, i is the map of input variables, o is the map of output variables, A is the set of activities, and L is the set of control flow connectors (control flow links).*

The set of activities A contains both basic and loop activities (cf. Definition 2 below). *Input variables* providing data to activities can be assigned using an input variable map $i : A \rightarrow \mathcal{P}(V)$. *Output variables* to which activities may write data to are described by the output variable map $o : A \rightarrow \mathcal{P}(V)$. Finally, the set of control flow connectors is $L \subseteq A \times A \times C$. A *control flow connector* $l \in L$ is a triple $l = (a_s, a_t, t \mid a_s, a_t \in A, t \in C \wedge a_s \neq a_t)$ connecting a source and a target activity, and its *transition condition* t (where C is the set of all conditions) is evaluated during run time. Note that the transition conditions allow to model typical workflow patterns such as parallel split and exclusive choice [11]. An activity $a \in A_*$ is called *start activity* if it is not the target of a control flow link: $A_* \subseteq A := \{a \mid a \in A \wedge \forall l \in L, a \neq \pi_2(l)\}$.[1]

Loops in the formal model for process models are expressed as complex activities that execute the loop as a sub-process according to a defined exit condition [10]. More formally:

Definition 2 (Loop Activity, a_l). *A loop activity $a_l \in A_L$ is a tuple $a_l = (m, A, L, \epsilon)$, where $m \in M$ is the name of the loop activity, A is a set of activities, L is a set of control flow connectors, and ϵ is a function $\epsilon : A_L \rightarrow C$ that assigns an exit condition to a loop activity.*

A loop activity is a container activity for other activities, regular (i.e., non-loop) as well as nested loop activities, and control flow connectors. The loop activity represents a do-until loop, which is executed at least once before the exit condition is evaluated. Our definition of a choreography model integrates the process model tuple as part of its participant definition:

Definition 3 (Choreography Model, $\mathfrak{C}$). *A choreography model is a directed, acyclic graph denoted by the tuple $\mathfrak{C} = (m, P, P_{set}, ML)$, where $m \in M$ is the name of the choreography model, P is the set of choreography participants, P_{set} is the set containing participant sets, ML is the set of message links between the choreography participants.*

A *choreography participant* $p \in P$ is a triple $p = (m, type, G)$, where $m \in M$ is the name of the participant, $type : P \rightarrow T$ is the function assigning a type $t_p \in T$ to the participant, and $G \in G_{all}$ is a process model graph, where G_{all} is the set of all process model graphs.

Typing the participant allows for several participants of the same type in the same choreography. Participants of the same type always possess the same process model graph. The set of all participants is denoted by P_{all}. A *participant set* $p_{set} \in P_{set}$ is described by $p_{set} \subseteq P_{all}$. This modeling construct is used to model a set of choreography participants whose number can be determined only during run time [12].

The set of message links ML is denoted as $ML \subseteq (P \cup P_{set}) \times P \times A \times A \times C$. A *message link* $ml \in ML$ is a tuple $ml = (p_s, p_r, a_s, a_r, t)$, where p_s, p_r are the sending and receiving participants, which must not be identical: $p_s \neq p_r$. This also holds for $a_s \in \pi_5(p_s.G)$ and $a_r \in \pi_5(p_r.G)$, which are the sending and receiving activities: $a_s \neq a_r$. The transition condition $t \in C$ is evaluated during run time.

2.2 Execution Phase

Choreography models are typically not directly executable [7], [13]. Instead, the refined process/workflow models implementing the choreography participants are instantiated. Together, they form an overall *virtual choreography instance*. The virtual choreography instance at any given point in time can be created by reading monitoring information, i.e., the execution traces of each process instance. We use the definitions of activity and process instance from [4] and extend them for choreography instances. Note that for

1. Note that we use the projection operator π_n to access the nth element of a tuple starting from index 1. $\mathcal{P}(X)$ denotes the power set of the set X including the empty set $\emptyset$. $p_y.X$ accesses element X (potentially a set) of element p_y.

our purposes it is sufficient to have a rather static notion of a choreography instance capturing only the accrued state at a certain point in time and not the advancement of the execution. This is due to the fact that the execution progress is suspended when we apply our algorithm, as we discuss in the following section.

Definition 4 (Process Instance, p_g). *An instance of a process model is a tuple $p_g = (V^I, A^A, A^F, L^E)$, where V^I is the set of variable instances, A^A is the set of active activity instances, A^F is the set of finished activity instances, and L^E is the set of evaluated links.*

For the set of variable instances it holds that: $V^I = \{(v, c, t) \mid v \in V, c \in DOM(v), t \in \mathbb{N}\}$. A variable instance provides a concrete value c from the domain of v ($DOM(v)$) for a variable v at a particular point in time t. The set of activity instances is defined as $A^I = \{(id, a, s, t, \sigma) \mid id \in ID, a \in A, s \in \mathcal{S}, t \in \mathbb{N}, \sigma \in \Sigma\}$, where ID is a set of unique identifiers, A is the set of activities, $\mathcal{S}$ is the set of states, $\mathbb{N}$ is the set of natural numbers indicating time, and Σ is the set of variable snapshot instances. A variable snapshot instance $\sigma \in \Sigma$ is defined as the triple $\sigma = (id, V^I_\sigma, t) \mid id \in ID, V^I_\sigma \subseteq V^I, t \in \mathbb{N}$.

The set $\mathcal{S} = \{S, E, C, F, T, Cmp, D\}$ contains the *execution states* an activity instance can take at any point in time t. Note that we describe states with the following abbreviations: (S = scheduled, E = executing, C = completed, F = faulted, T = terminated, Cmp = compensated, D = dead, Sus = suspended). *Completed* activity instance have been successfully carried out their intended work, while *terminated* ones were forcefully aborted during execution.

The state of an activity instance $a^i \in A^I$ can be determined by the function $state(a^i)$, whereas its model element is retrieved by the function $model(a^i)$. The set of activity instances A^I contains both non-loop and loop (cf. Definition 6) activity instances. For the set of active activity instances the following holds: $A^A \subseteq A^I, \forall a^i \in A^A : state(a^i) \in \{S, E\}$. If an activity is executed in a loop, a new instance is created for each iteration, i.e., there is at most one activity instance a^i of an activity a in the set of active activities A^A. For the set of finished activity instances the following holds: $A^F \subseteq A^I, \forall a^i \in A^F : state(a^i) \in \{C, F, T, D\}$. *Compensated activity instances* (state = Cmp) are not in the set A^F because their effects have been semantically undone. For the set of *evaluated control flow links* the following holds: $L^E = \{(l, c, t) \mid l \in L, c \in \{true, false\}, t \in \mathbb{N}\}$. Evaluated links already have an truth value c assigned at time t determining if the link has been followed during execution. Furthermore:

Definition 5 (Instance Subgraph, g^i). *An* instance subgraph *is a directed, acyclic graph represented by a tuple $g^i = (V^I, A^A, A^F, L^E)$, where $g^i \preceq_{g^i} p_g$.*

The $\preceq_{g^i}$ operator means that the elements of the tuple g^i are a subset or equal to the corresponding elements in the process instance tuple p_g. We include the instance subgraph definition in order be able to algorithmically handle the process graph and any subgraphs of it in the same way.

Definition 6 (Loop Activity Instance, a^i_l). *A* loop activity instance *is a tuple $a^i_l = (id, a_l, A^A, A^F, L^E, ctr, s, t)$, where $id \in ID$ is a unique identifier of the instance, a_l is the loop activity, A^A is the set of active activity instances, A^F is the set of finished activity instances, L^E is a set of evaluated control flow links, $ctr \in \mathbb{N}$ is the loop counter of the loop activity instance, $s \in \mathcal{S}$ is the current state of the instance, $t \in \mathbb{N}$ is the instance's execution time.*

It then follows:

Definition 7 (Choreography Instance, $\mathbf{c}^i$). *A choreography instance is the pair $\mathbf{c}^i = (P^I, ML^E)$, where P^I is the set of participant instances belonging to the choreography instance and ML^E the set of evaluated message links.*

The set of participant instances P^I contains pairs of the form $p^i = (m, p_g)$, where m is the name of the participant instance and $p_g \in P^{all}_g$ is a process instance. For ML^E the following holds: $ML^E = \{(ml, c, t) \mid ml \in ML, c \in \{true, false\}, t \in \mathbb{N}\}$, i.e., ML^E contains the instantiated message links having a truth value c indicating the outcome of the transition condition evaluation at execution time t.

3 Rewinding & Repeating Choreographies

In this section we present our overall approach for rewinding and repeating choreography logic in a choreography instance. We introduce the method our approach follows and the algorithm for automatic identification of rewinding points.

3.1 Prerequisites

A basic assumption for the *Model-as-you-go for Choreographies* approach is the existence of a monitoring infrastructure capturing the execution events and providing information about instance states of the process models distributed across different execution engines. These states are correlated with the corresponding choreography model, both in the supporting middleware and in the graphical modeling and monitoring environment a scientist uses. Based on the life cycle for scientific choreographies, which we introduced in [13], it is possible to switch between the levels of choreography and workflows during execution for monitoring purposes on different degrees of detail and to perform adaptation actions as well starting the repetition on both levels. Each scientist has access to a common choreography model and at least to the workflow models she has refined by herself. This includes the monitoring of the execution state. Access to all other refined workflow models and the execution events depends on the access rights given to the particular scientist by the owner of the workflow model. Furthermore, scientists must be able to control the execution of the choreography instance. Choreography instance control entails the starting, suspending, resuming, and terminating of participating workflow instances in a coordinated fashion in order to be enable to react to deviations and errors, and to perform adaptation actions.

3.2 Concepts & Method

One way to enable reaction to deviations and errors is through repeating workflow/experiment logic using either of two operations we defined for individual (scientific) workflows. *Iteration* is the repetition of parts of the logic in a workflow instance without undoing already completed work. The executed activities and links in the engine internal representation are simply reset; note that historical data

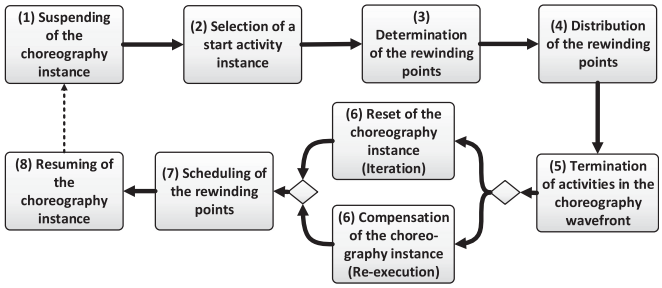


Fig. 2. Method for rewinding and repetition of choreography logic.

are kept and past state changes are not overwritten. Optionally, the scientist may decide to load stored variable snapshots for the re-run and not use the current values. Basically, iteration resembles the execution semantics of a loop construct without having it explicitly modeled. *Re-execution* denotes the operation of repeating workflow logic in a single workflow instance after undoing the already completed work by invoking compensation activities in the reverse order for executed service invocations, resetting control flow links, and by loading stored variable snapshots for the start activity [4]. In the context of (scientific) choreographies, we analogously define *iteration in a choreography instance* as the repetition of logic in the enacting workflow instances without undoing already completed work. The workflow instances participating in the choreography instance are collectively reset. *Re-execution in a choreography instance* is the repeat of logic after compensating already completed work. That means the choreography instance is reset to a state of its past. The involved workflow instances have however to be compensated collectively. The rewinding and repetition of choreography logic can be conducted by adhering to the steps of the method depicted in Fig. 2. The steps involved are explained in the following sections making use of several conceptual figures[2] (Figs. 3, 4, and 5).

3.2.1 Step 1: Suspending of the Choreography Instance

This is done either by a scientist or by the workflow engines reaching the end of the defined workflow models. The user-initiated suspend has to be conducted in a coordinated fashion with all relevant workflow engines by using reliable messaging [14] and/or transactional concepts [15]. It implies that no further execution progress in any of the involved workflow instances occurs until they are collectively resumed again (cf. Step 8).

3.2.2 Step 2: Selection of a Start Activity Instance

Repeating parts of the logic of choreography instances is triggered by the scientist choosing a *start activity* in a *start participant instance* (activity c_1 of $Participant_1$ in Fig. 3) during run time via a graphical modeling and monitoring environment. As multiple (completed) instances for one model element might exist, especially inside loop activities, the modeling and monitoring environment must support a

2. In the following, the subscript of an element denotes a part of the element name, while the superscript indicates the instance id. However, we only show the instance id if there are more than one instances of the same element.

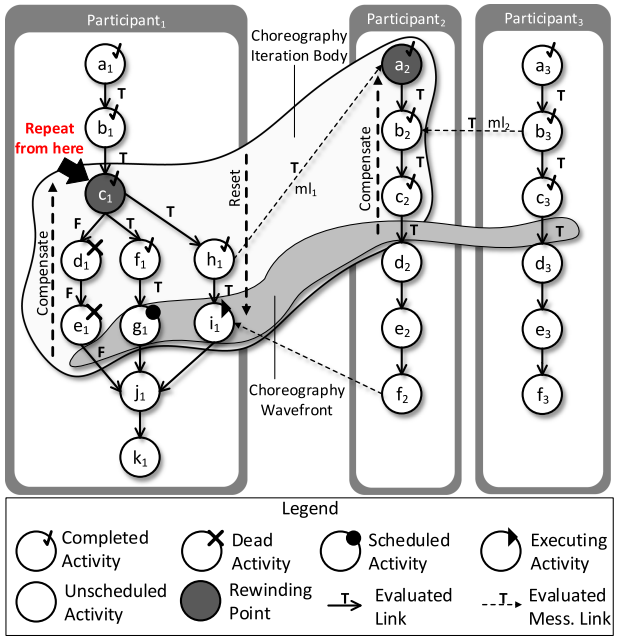


Fig. 3. Example of a choreography instance without loops.

convenient selection, for example by using a graphical wizard that leads the user through the selection step by step. The selection can either take place on the level of the choreography or on the workflow level depending on the access rights of the scientist.

When monitoring the scientific choreography instance, *data*, i.e., the variable values, of the workflow instances is one factor scientists have to consider for the repetition of logic. Similar to database logs for recovery [16], in our existing work [4] we log variable changing events into stable storage. A snapshot-enabled workflow engine stores the *snapshot instances* with every variable-changing activity in its database. Moreover, it offers a service interface to the outside to retrieve the snapshot instances related to any given activity. A snapshot instance contains references to all variables visible for a particular activity, the current variable values, and a creation timestamp. Fig. 4 shows the simplest (i.e., sequential) selection case in a workflow instance. If it is to be repeated from activity instance c, the snapshot instance attached to activity instance b has to be loaded. It is the nearest snapshot instance, i.e., has the most recent timestamp ($t = 2$), on the execution path to activity instance c. The selection of snapshot instances from parallel execution paths are also considered in [4], however, we do not recapitulate these concepts here due to space limitations.

We base our data handling on this concept and allow the user to select either manually or automatically the stored snapshot instance with the most recent timestamp for the start activity instance. This means the corresponding

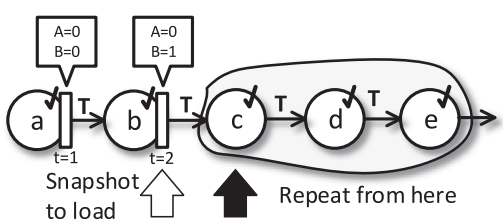


Fig. 4. Data handling in the sequential case. Adapted from [4].

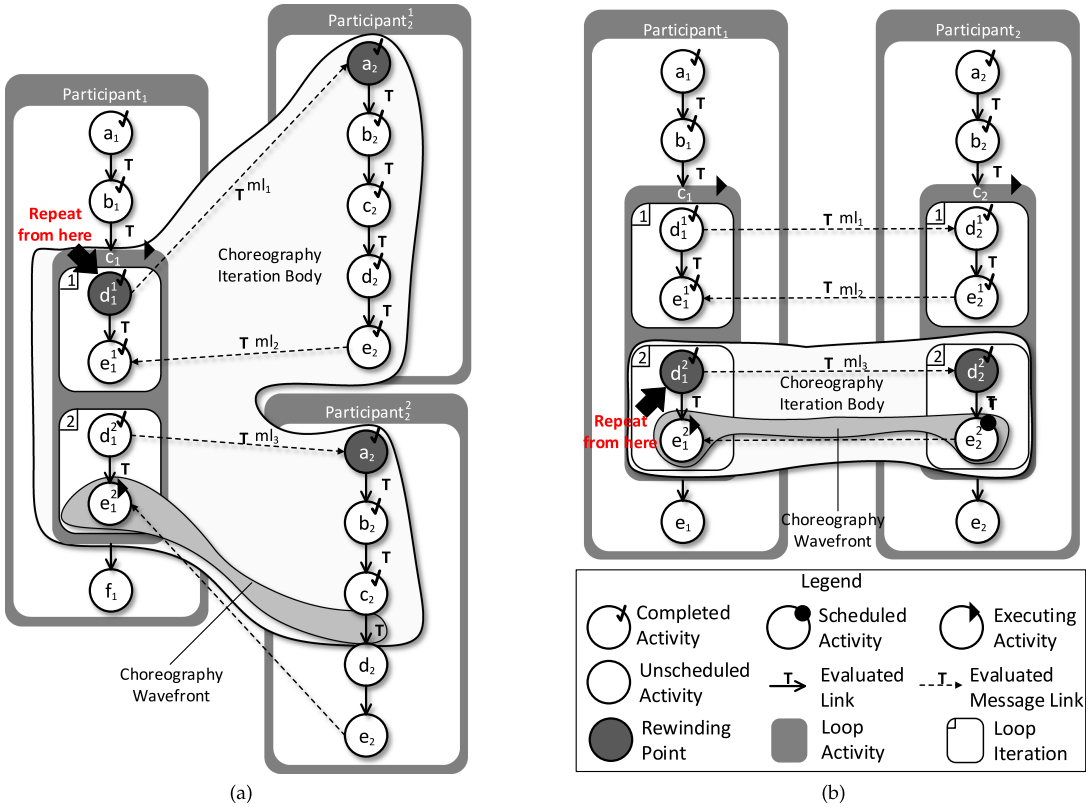


Fig. 5. Loop examples. (a): Choreography instance with loops and instantiated participant set. (b): Choreography instance with synchronizing loops.

workflow engine is queried for all snapshot instances related to the selected start activity instance and the results are presented to the user in a graphical manner. The option is provided for both iteration and re-execution. In case of re-execution, the compensation logic is only responsible for undoing state which is external to the participant. Therefore, a snapshot instance is has to be selected and loaded to provide the desired variable values, which are part of the internal state of the workflow instance, before starting the re-execution. In case of iteration it is optional.

The refinement of the choreography participants might introduce variables which are not part of the choreography model. Two cases can be distinguished: (i) A scientist wishes to start the repetition of a choreography instance from the choreography or workflow level where he/she has access to the refined workflow model. In this case, graphical facilities are provided to select the desired snapshot instance and all or a subset of the variables visible for the start activity instance. (ii) A scientist wishes to start the repetition on the level of the choreography from a participant instance where he/she does not have access to the refined workflow. Here, the manual selection of snapshots should not be offered. Instead, the engine internally uses the most recent variable snapshot instance preceding the start activity instance from the corresponding workflow engine's database as shown in Fig. 4. The same approach is employed for all calculated rewinding points outside the start participant instance.

3.2.3 Step 3: Determination of the Rewinding Points

An important concept is the notion of *iteration body*. In [4], the iteration body is defined as the instantiated activities and evaluated links reachable from a user-selected start activity instance of a single process instance. In Fig. 3, the iteration body of $Participant_1$ consists of the activities c_1, d_1, e_1, f_1, g_1, h_1, i_1, and the control flow connectors between them. Note that the path $c_1 \rightarrow d_1$ has not been chosen during execution and is marked as *dead*. However, it may be included into the iteration body when starting a repetition from activity c_1. In the context of choreographies, the iteration body spans across process instances and includes message link instances between them. In Fig. 3, in addition to the already enumerated ones, the activities a_2, b_2, c_2, the control flow links between them as well as the control flow link $c_2 \rightarrow d_2$, and the message link instance ml_1 are part of the *choreography iteration body*. In other words, the choreography iteration body contains all activity, control flow, and message link instances reachable from the start activity instance. The repetition of logic starting in one particular participant instance affects at least all participant instances that are part of the choreography iteration body.

Here, two cases can be distinguished. In the first case, the start participant instance is connected, directly or transitively, to other participants that are reachable from the manually selected start activity instance (c_1 in Fig. 3). That means, the start participant instance contains *completed* sending activity instances that have sent messages to other participant instances which themselves may have invoked further ones. Already completed activity instances on the execution path must be *rewound*, i.e., either be reset (iteration) or compensated (re-execution) across the affected choreography participant instances. While the start activity instance for the repetition is simultaneously the so-called *rewinding point* in the start participant instance, the rewinding points in the

connected instances have to be identified separately. Each rewinding point indicates where the resetting or compensation of activities has to be stopped. In the example, activity a_2 is the rewinding point of $Participant_2$.

In the second case, participant instances that are not reachable from the start activity instance may still be affected by the repetition of logic in other participant instances. Messages that are not the reply to previous requests and have been transmitted over incoming message links to the affected participant instances must be available again in the case of repetition. This can be done by storing and replaying previously sent messages by the workflow engine or a middleware component responsible for the respective participant instance. An example for this case are $Participant_3$ and the message link instance ml_2 in Fig. 3.

The concept for determining the rewinding points in choreography instances is refined in this work to also consider *multiple instances of the same participant*, i.e., instantiated participant sets (cf. Definition 3), as well as *loop activity instances* as part of the participant instances. Fig. 5a shows an example of a choreography instance containing both loop activity instances and two instances of a choreography participant modeled with the participant set construct. In order to find the rewinding points in all participant instances, all iterations of an instantiated loop activity have to be traversed. The iterations of a loop activity instance can be seen as independent instance subgraphs (cf. Definition 5) where each has to be traversed sequentially. We call this the *loop instance graph* of a particular loop iteration. In the example of Fig. 5a, this means that starting from the initial start activity instance d_1^1 in the first iteration of loop activity instance c_1, the participant instance $Participant_2^1$ is reached via the message link instance ml_1. Activity instance a_2 is the rewinding point of $Participant_2^1$. After finishing the traversal of $Participant_2^1$ and the loop activity instance iteration 1, the loop instance graph of loop activity instance iteration 2 is traversed. Here, $Participant_2^2$ is reached via the message link instance ml_3 and activity instance a_2 can be marked as rewinding point.

Our approach supports two more cases, both shown in Fig. 5b. First, we allow users to select an activity instance in an arbitrary iteration of an loop activity instance. In the example in Fig. 5b this is the activity instance d_1^2 located in iteration 2 of loop activity instance c_1. Second, both participant instances possess loops that synchronize with each other, e.g., participant instance $Participant_2$ also possesses a loop activity instance (c_2). Following the message link instance ml_3 to find the rewinding point in $Participant_2$, the correct loop iteration of c_2 must be entered to mark the activity instance d_2^2 as a another rewinding point.

In order to automatically determine the rewinding points, we introduce an algorithm in Section 3.3. The data structure to store the rewinding points of a choreography instance is defined in the following way:

Definition 8 (Choreography Rewinding Points, $RP_{\mathfrak{C}}$). *The data structure* Choreography Rewinding Points $RP_{\mathfrak{C}} \subseteq P^I \times \mathcal{P}(A^I)$ *is a set of pairs* $\{(p^i, A^I_{rp}) \,|\, p^i \in P^I, A^I_{rp} = \{a^i_1, \ldots, a^i_k\} \subseteq A^I\}$ *consisting of a participant instance and a set of rewinding point activity instances.*

The reason that a participant instance can have more than one rewinding point is the existence of parallel paths in the process model graph. A participant may receive messages in parallel that result in independent rewinding points.

3.2.4 Step 4: Distribution of the Rewinding Points

In this step, the automatically determined rewinding points as well as the optionally selected variable snapshot instance (of the start participant instance) have to be distributed to the workflow engines that host the involved workflow instances. It has to be ensured that the rewinding points reach all relevant workflow engines, e.g., by using reliable message-oriented middleware [14] and/or transactional concepts [15].

3.2.5 Step 5: Termination of Activities in the Choreography Wavefront

Manually triggered repetition of logic on an execution path can lead to race conditions in individual workflows [4] as well as in choreographies. Race conditions include the execution of one path in parallel inside one process or choreography instance. To deal with this issue the activities of the *choreography wavefront* have to be terminated. The choreography wavefront contains all currently active instances or scheduled elements such as activities, control flow connectors, and message links. In Fig. 3, for example, activities g_1 and i_1 of $Participant_1$ are currently running or are scheduled, respectively, and will subsequently trigger the competing execution of activity j_1 if not terminated before starting the repetition from activity instance c_1. The termination is handled locally in each involved workflow instance after receiving the rewinding points.

3.2.6 Step 6: Rewinding the Choreography Instances

The rewinding resets the choreography iteration body to a previous state and moves the choreography wavefront to the past. This has to be conducted locally by each involved workflow engine. However, different approaches are used for iteration and re-execution (cf. Fig. 2). For the rewinding of each individual workflow instance we reuse the concepts of [4]. That means in the case of iteration that the activities and links in the iteration body of each participant instance are reset by the corresponding workflow engine to enable a subsequent execution. See Fig. 3 for an example that shows the resetting direction from the rewinding point until reaching the choreography wavefront. Optionally, data snapshots are automatically determined for the rewinding points (cf. Step 2).

We formally define the iterate operation of choreography logic in the following way:

Definition 9 (Iterate operation, $\mathfrak{i}_c$). *The* iterate operation for choreographies *is defined as the function* $\mathfrak{i}_c : A^I \times \Sigma \times RP^{all}_{\mathfrak{C}} \times \mathfrak{C}^I \rightarrow \mathfrak{C}^I$, *where* A^I *is the set of activity instances,* Σ *is the set of variable snapshot instances,* $RP^{all}_{\mathfrak{C}}$ *is the set of all choreography rewinding points (cf. Definition 8), and* $\mathfrak{C}^I$ *is the set of choreography instances.*

The iterate operation takes as input the start activity instance $a^i_{start} \in A^I$, the corresponding variable snapshot instance $\sigma \in \Sigma$ of a^i_{start}, the set of determined rewinding points assigned to their corresponding participant instances $RP_{\mathfrak{C}} \in RP^{all}_{\mathfrak{C}}$, and the affected choreography instance $\mathfrak{c}^i \in \mathfrak{C}^I$.

The re-execution operation is an extension of iteration, where additionally to the resetting of control flow connectors, the iteration body of each participant instance is

compensated in reverse order (cf. Fig. 3) and a data snapshot is automatically determined for the rewinding points in each involved workflow engine. The semantics of the input parameters is identical to the iterate operation. Note that the re-execution operation is only applicable if corresponding compensation services have been implemented for a particular use case. We also extended the traversal of the iteration body by the workflow engines for both operations to also consider multiple rewinding points.

A special case to be considered is the complete rewinding of a participant instance. This occurs when the model of the rewinding point activity instance is an instance creating activity (for example activity instance a_2 of $Participant_2$ in Fig. 3). Instead of keeping the participant instance alive, it should be terminated since it cannot be guaranteed whether an instance of that particular participant is needed again during the repetition. The new execution might take a completely different path. However, simply terminating the participant instance is not sufficient. In case of re-execution the iteration body still has to be compensated to undo its effects. In case of iteration, the iteration body has to be reset and the variable values have to be stored. If the repeated execution takes the same path, the variable values have to be loaded for a new participant instance of the same type. However, if the participant instance is created on a different workflow engine, the variable values have to be migrated.

3.2.7 Step 7: Scheduling of the Rewinding Points

The rewinding points are scheduled in their respective workflow engines to be executed next. An exception are the rewinding points that are located in participant instances that have been terminated after rewinding.

3.2.8 Step 8: Resuming of the Choreography Instance

The last step entails the resuming of the execution of the choreography instance. To each participant instance, with the exception of the completely rewound and terminated ones, a message is sent to resume execution.

The method's steps can be repeated if a scientist recognizes the need for further repetitions. Here, no new challenges for the determining of the rewinding points arise. Furthermore, if one step of the method fails, all steps apart from the compensation of activities in case of re-execution can be easily retried by starting the method anew. This is due to fact that the workflow engines keep their instance state in stable storage and the calculation of the rewinding points and resetting already reset activities does not do any damage. However, retrying the compensation step would need a idempotent implementation of external services providing the compensation functionality.

3.3 Determining the Rewinding Points

In the following, we present an algorithm to automatically determine the rewinding points in a choreography iteration body, that realizes Step 3 of the proposed method. The main idea of the algorithm is to traverse the choreography instance graph beginning from the user-selected start activity instance in the start participant instance, follow the executed message link instances and thus identify all choreography participant instances that are part of the choreography iteration body. In doing so, the algorithm collects all rewinding points. The algorithm is divided into four parts and supported by a set of auxiliary functions, which are defined and explained as they occur in the algorithm. The rewinding point algorithm realizes the function ρ.

Definition 10 (Function ρ). *The* Determine Rewinding Points function ρ *is defined as* $\rho : \mathfrak{C}^I \times A^I \times P^I \times RP^{all}_{\mathfrak{C}} \rightarrow RP^{all}_{\mathfrak{C}}$, *where* $\mathfrak{C}^I$ *is the set of choreography instances,* A^I *is the set of activity instances,* P^I *is the set of participant instances, and* $RP^{all}_{\mathfrak{C}}$ *is the set that contains all* $RP_{\mathfrak{C}}$ *sets (cf. Definition 8).*

$RP_{\mathfrak{C}}$ is the data structure that contains the determined rewinding points for each involved participant instance. Before the first invocation of ρ the data structure $RP_{\mathfrak{C}}$ is empty: $RP_{\mathfrak{C}} = \emptyset$. Algorithm 1 is the starting point for the automatic determination of the rewinding points. The initial invocation of the algorithm needs the start participant instance and the user-selected start activity instance. First, it is checked if the start activity instance a^i_{start} is nested inside a loop activity instance using the *getEnclosingLoop* function (cf. Definition 11 below). If so, the user has selected an activity instance inside an iteration of a loop activity instance and the sub-routine λ (*handleLoopActivity*) defined in Definition 15 and realized by Algorithm 4 is directly invoked. Otherwise, a^i_{start} is not nested inside a loop and the function τ (*traverseInstanceSubgraph*) is invoked to traverse the process instance graph p_g of p^i.

Algorithm 1. determineRewindingPoints, ρ

1 **input:** Choreography instance $\mathfrak{c}^i$, activity instance a^i_{start}, participant instance p^i, set of pairs $RP_{\mathfrak{C}} = (p^i, A^I_{rp})$
2 **output:** $RP_{\mathfrak{C}}$
3 **begin**
4 **if** $RP_{\mathfrak{C}} = \emptyset$ **then**
5 $RP_{\mathfrak{C}} \leftarrow (p^i, \{a^i_{start}\})$
6 **end**
7 Loop Activity Instance $a^i_l \leftarrow$ `getEnclosingLoop`(a^i_{start})
8 **if** $a^i_l = \perp$ **then**
9 $\tau\,(c^i, p^i, p^i.p_g, a^i_{start}, RP_{\mathfrak{C}})$
10 **else**
11 $\lambda(\mathfrak{c}^i, p^i, a^i_l, a^i_{start}, RP_{\mathfrak{C}})$
12 **end**
13 **return** $RP_{\mathfrak{C}}$
14 **end**

Definition 11 (Function getEnclosingLoop). *The* getEnclosingLoop function *is defined as* $getEnclosingLoop : A^I \rightarrow A^I_L$, *where* A^I *is the set of activity instances and* A^I_L *is the set of loop activity instances.*

The *getEnclosingLoop* function is used to determine if there is a loop activity instance $a^i_l \in A^I_L$ enclosing a given activity instance $a^i \in A^I$. It returns $\perp$ if a^i does not have a parent loop activity instance, and function τ is invoked:

Definition 12 (Function τ). *The* Traverse Instance Subgraph τ function *is defined as* $\tau : \mathfrak{C}^I \times P^I \times G^I \times A^I \times RP^{all}_{\mathfrak{C}} \rightarrow RP^{all}_{\mathfrak{C}}$, *where* $\mathfrak{C}^I$ *is the set of choreography instances,* P^I *is the set of participant instances,* G^I *is the set of instance subgraphs,* A^I *is the set of activity instances, and* $RP^{all}_{\mathfrak{C}}$ *is the set that contains all* $RP_{\mathfrak{C}}$ *sets (cf. Definition 8).*

The function τ is used to traverse an instance (sub-)graph. It is realized by Algorithm 2, the main idea being the following: beginning from the activity instance a^i_{start}, the instance subgraph g^i is traversed in a depth-first manner. g^i can be the complete process instance graph p_g or a loop instance graph of some iteration of a loop activity instance. For each activity instance a^i it is checked if it is a completed sending activity instance (line 10). If so, the sub-routine χ (*handleSendingActivity*) as defined in Definition 13 is invoked (line 11). Subsequently, the outgoing control flow connectors of every completed activity instance are followed by pushing them on the stack data structure, provided they have been evaluated to true (lines 12-14). If the model of a^i ($model(a^i)$) is a loop activity, the sub-routine λ (*handleLoopActivity*, cf. Definition 15 and Algorithm 4) is invoked instead (lines 15-17).

Algorithm 2. traverseInstanceSubgraph, τ

```
1  input: Choreography instance c^i, participant instance p^i,
       instance subgraph g^i, activity instance a^i_start, set of
       pairs RP_c = (p^i, A^I_rp)
2  output: RP_c
3  begin
4    Stack S ← ∅
5    S.push(a^i_start)
6    while S ≠ ∅ do
7      a^i ← S.pop()
8      if a^i is not marked as visited then
9        mark a^i as visited
10       if model(a^i) is sending activity ∧
         state(a^i) = completed then
11         RP_c ← χ(c^i, a^i, RP_c)
12         foreach l^i = (l^x, c^x, t^x) ∈ g^i.L^E | l^x.a^x_s =
           a^i ∧ state(c^x) = true do
13           S.push(l^x.a^x_t)
14         end
15       else if model(a^i) is loop activity then
16         RP_c ← λ(c^i, p^i, a^i, ⊥, RP_c)
17       end
18     end
19   end
20   return RP_c
21 end
```

Definition 13 (Function χ). *The* Handle Sending Activity function χ *is defined as* $\chi : \mathfrak{C}^I \times A^I \times RP^{all}_{\mathfrak{C}} \rightarrow RP^{all}_{\mathfrak{C}}$, *where* $\mathfrak{C}^I$ *is the set of choreography instances,* A^I *is the set of activity instances, and* $RP^{all}_{\mathfrak{C}}$ *is the set of pairs containing the assignment of participant instances to their rewinding points (*$RP_{\mathfrak{C}}$, *cf. Definition 8).*

Algorithm 3 (handleSendingActivity) realizes the function χ. First, the message link instance $ml^i_{traversed}$, which is attached to the sending activity instance a^i, is retrieved by evaluating the following conditions: (i) it has been evaluated to true and (ii) there is a *receiving* activity instance a^i_r in the *completed* state, i.e., a message has been sent and consumed (line 4). We assume reliable FIFO channels for communication, i.e., all messages in transit have reached their destination before we conduct any rewinding. If $ml^i_{traversed}$ is not empty, the algorithm follows the message link instance and retrieves the receiving participant instance (lines 5-6). By exactly identifying the receiving participant instance it is also possible to consider instances which were modeled by a participant set. For the receiving participant instance it is checked if it has already been (partly) traversed by the algorithm and a (preliminary) rewinding point has been found. If this is not the case, the receiving activity instance a^i_r is added as a rewinding point for the receiving participant p^i_r and ρ is invoked recursively using p^i_r as input (lines 7-9). If there exists already a rewinding point for p^i_r (line 10), it is checked if (i) the old rewinding point would be a successor of the new one (using the $succ$ function defined in Definition 14 below) or if (ii) both are in parallel branches. In case (i) the old rewinding point activity instance is removed before the new rewinding point is added and in case (ii) both are kept (lines 11-19). In both cases, ρ is invoked recursively afterwards (lines 20-23). The recursion in one participant instance stops when all reachable completed activity instances have been marked as visited.

Algorithm 3. handleSendingActivity, χ

```
1  input: Choreography instance c^i, activity instance a^i, set of
       pairs RP_c = (p^i, A^I_rp)
2  output: RP_c
3  begin
4    Message Link Instance ml^i_traversed ← (ml^x, c^x, t^x) | (ml^x,
     c^x, t^x) ∈ ML^E ∧ ml^x = (p^i_s, p^i_r, a^i_s, a^i_r, c) ∧ a^i = a^i_s ∧ state(c^x) =
     true ∧ state(a^i_r) = completed
5    if ml^i_traversed ≠⊥ then
6      Participant Instance p^i_r ← ml^i_traversed·p^i_r
7      if ∄(p^x, A^x_rp) ∈ RP_c | p^x = p^i_r then
8        RP_c ← RP_c ∪ (p^i_r, {a^i_r})
9        RP_c ← ρ(c^i, a^i_r, p^i_r, RP_c)
10     else if ∃(p^x, A^x_rp) ∈ RP_c | p^x = p^i_r then
11       Boolean recursion ← false
12       foreach a^x ∈ A^x_rp do
13         if succ(a^i_r, a^x) then
14           A^x_rp ← A^x_rp \ a^x ;
15           recursion ← true ;
16         else if ¬succ(a^x, a^i_r) ∧ ¬succ(a^i_r, a^x) then
17           recursion ← true ;
18         end
19       end
20       if recursion then
21         A^x_rp ← A^x_rp ∪ a^i_r ;
22         RP_c ← ρ(c^i, a^i_r, p^i_r, RP_c)
23       end
24     end
25   end
26   return RP_c
27 end
```

Definition 14 (Function succ). *The* successor function $succ$ *is defined as* $succ : A^I \times A^I \rightarrow \mathbb{B}$, *where* A^I *is the set of activity instances and* $\mathbb{B}$ *is the set of boolean values* $\mathbb{B} = \{true, false\}$.

The function determines if the second activity instance is reachable from the first activity instance, and thus is a successor in the process instance graph. When determining the successor property, the iterations of loop activity instances must also be considered.

The handling of loop activity instances and their iterations is introduced using the λ function.

Definition 15 (Function λ). *The* Handle Loop Activity *function λ is defined as $\lambda : \mathfrak{C}^I \times P^I \times A_L^I \times A^I \times RP_{\mathfrak{C}}^{all} \rightarrow RP_{\mathfrak{C}}^{all}$, where P^I is the set of participant instances, A_L^I is the set of loop activity instances, A^I is the set of activity instances, and $RP_{\mathfrak{C}}^{all}$ is the set of pairs containing the assignment of participant instances to their rewinding points ($RP_{\mathfrak{C}}$, cf. Definition 8).*

Definition 16 (Function getLoopIteration). *The* getLoopIteration *function is defined as $getLoopIteration : A^I \rightarrow \mathbb{N}$, where A^I is the set of activity instances and $\mathbb{N}$ is the set of natural numbers.*

This function is used to determine the loop iteration of the loop activity instance where a particular activity instance $a^i \in A^I$ is located in.

Definition 17 (Function getLoopInstanceGraph). *The* getLoopInstanceGraph *function is defined as $getLoopInstanceGraph : A_L^I \times \mathbb{N} \rightarrow G^I$, where A_L^I is the set of loop activity instances, $\mathbb{N}$ is the set of natural numbers, and G^I is the set of instance subgraphs.*

This function is used to retrieve the instance subgraph $g^i \in G^I$ corresponding to a particular loop iteration $ctr \in \mathbb{N}$ of an loop activity instance $a_l^i \in A_L^I$.

The λ function's realization (as required by Algorithm 1) is introduced in Algorithm 4. If there exists a start activity instance $a_{start}^i \in a_l^i.A^A \cup a_l^i.A^F$ nested in the loop activity instance a_l^i, the iteration number $iter_a$ is retrieved using the *getLoopIteration* (cf. Definition 16) function (lines 6-9). Otherwise, $iter_a$ has been initialized to 1 and all loop iterations are traversed. The algorithm iterates while the currently iteration $iter_{curr}$ is smaller or equal to the overall number of iterations of a_l^i indicated by the loop counter $a_l^i.ctr$ (line 10). For each executed iteration, the instance subgraph of the current loop iteration is retrieved using the *getLoopInstanceGraph* (cf. Definition 17) function (line 11). If a_{start}^i exists and it is the first traversal of the loop activity instance ($iter_a = iter_{curr}$), the sub-routine τ (traverseInstanceGraph) is directly called. That means, the traversal does not start at the beginning of the loop instance graph, but rather from a_{start}^i (lines 12-13). Otherwise, the traversal comprises the complete loop instance graph g^i and τ is called for each activity instance a^i, which is a start activity, i.e., $model(a^i) \in A_*$ (cf. Definition 1) (lines 15-17). After the traversal of g^i, $iter_{curr}$ is incremented by 1.

4 Realization

In the following, we show how our approach for rewinding and repeating of choreography logic is realized in our *ChorSystem*. The service-oriented and message-based ChorSystem enables users to manage the complete life cycle of choreographies from modeling to execution [17]. The life cycle starts with choreography modeling, transformation to and refinement of workflow models. The refined workflow models are distributed among a set of workflow engines in an automated manner while a logical representation of the choreography is created in the so-called *ChorSystem Middleware*. When starting a new choreography instance, a corresponding representation is created and updated by monitoring execution events. The life cycle management operations for suspending, resuming, and terminating act upon this representation. The functionality of the middleware is defined by composing the different components using message-based Enterprise Integration Patterns [14]. The generic architecture and the control and deployment aspects of choreography life cycle management have been initially introduced in [17]. Due to space limitations we will not discuss the complete system architecture and functionalities in detail and will restrict ourselves to showing with the help of Fig. 6 how the rewinding and repetition method (cf. Fig. 2) is supported by the ChorSystem.

Algorithm 4. handleLoopActivity, λ

```
1  input: Choreography instance c^i, participant instance p^i,
         activity loop instance a_l^i, activity instance a_start^i, set of
         pairs RP_c = (p^i, A_rp^I)
2  output: RP_c
3  begin
4      Number iter_curr ∈ ℕ ← 1
5      Number iter_a ∈ ℕ ← 1
6      if a_start^i ≠⊥ then
7          iter_a ← getLoopIteration(a_start^i)
8          iter_curr ← iter_a
9      end
10     while iter_curr ≤ a_l^i.ctr do
11         Instance Subgraph
           g^i ← getLoopInstanceGraph(a_l^i, iter_curr)
12         if a_start^i ≠⊥ ∧ iter_a = iter_curr then
13             RP_c ← τ(c^i, p^i, g^i, a_start^i, RP_c)
14         else
15             foreach a^i ∈ g^i.A^F | model(a^i) ∈ A_* do
16                 RP_c ← τ(c^i, p^i, g^i, a^i, RP_c)
17             end
18         end
19         iter_curr ← iter_curr + 1
20     end
21     return RP_c
22 end
```

In the *ChorDesigner*, which acts as integrated modeling and monitoring environment as well as control panel for the life cycle management operations, the user gives the command to suspend a running choreography instance (Method step 1). The suspend command is processed by the *Instance Manager* in the so-called *Control Route* [17]. To realize the repetitions the Instance Manager carries out the *Repetition Route* depicted in Fig. 6. The selection of the start activity instance and the corresponding variable snapshot (step 2) are conducted together with the ChorDesigner. This information is sent via messaging to the *Instance manager* where the repetition functionality is found using the *Content-Based Router* pattern. Step 3 of the method is realized by retrieving a choreography instance representation from the Event Registry using a *Custom Message Processor* and subsequently calculating the rewinding point in the second *Custom Message Processor* implementing the Algorithm introduced in Section 3.3. In step 4 with the help of the *Recipient List* pattern, the affected workflow engines are

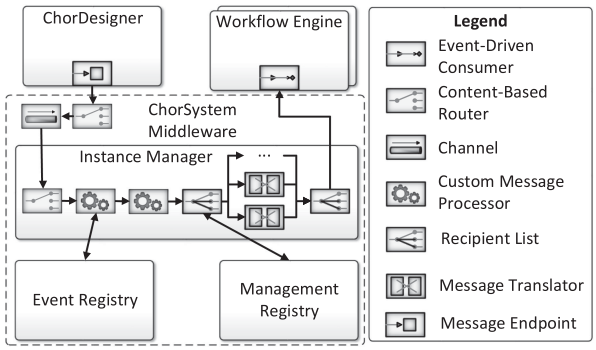


Fig. 6. Repetition route described by the enterprise integration patterns.

determined by accessing the *Management Registry* and translating the repetition message into the workflow engine specific format and adding the calculated rewinding points. The messages are then fanned out to the workflow engines. The reset or compensation of the involved workflow instances (step 6) and the scheduling of the rewinding points (step 7) are carried out in the respective workflow engines as described in Section 3 and in [4]. Resuming execution of the choreography instance is again triggered by the user in the ChorDesigner.

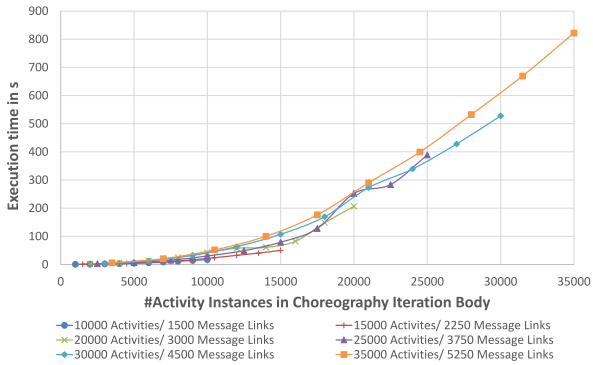
We have implemented the architecture and our approach using open source software and standards. We employ BPEL4Chor [18] as choreography language and BPEL [19] as executable workflow language. The ChorDesigner and the ProcessDesigner are based on Eclipse[3] technologies, while for the ESB Apache ServiceMix[4] is employed. The workflow engines are based on an extended Apache ODE,[5] and the message routes to coordinate the middleware components are realized with Apache Camel.[6] The algorithm for finding the rewinding points is implemented in Java. The ChorSystem Middleware's source code is available on GitHub.[7] Detailed information on the ChorSystem and its architecture can be accessed online on our project website [20].

5 Evaluation

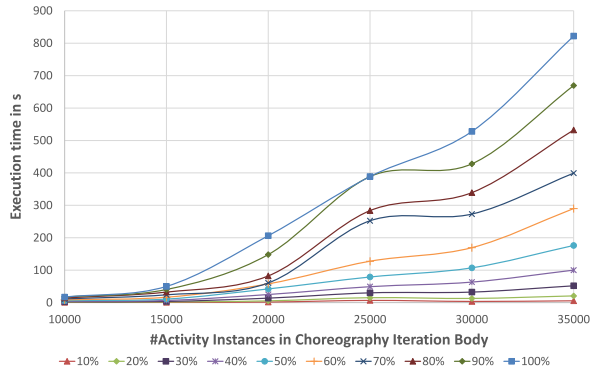
In this section, we evaluate the performance of the proposed algorithm for the rewinding of choreography instances in terms of execution time. The focus in this article on the algorithmical aspect only is due to space reasons. The interested reader is referred to the evaluation of our prototypical ChorSystem by means of an case study, which is available online [20]. All measurements were conducted on an Ubuntu 14.04 LTS virtual machine equipped with 1 CPU core (2.29 GHz) and 4 GB RAM and represent the median values of 5 runs. The evaluation consists of three major parts.

First, we generated (randomly) 6 choreography models having 10 participants each. The reason for generating artificial models is that the execution of our motivating example (cf. Fig. 1) yields execution times of the rewinding

3. http://eclipse.org/modeling
4. http://servicemix.apache.org
5. http://www.iaas.uni-stuttgart.de/forschung/projects/ODE-PGF/
6. http://camel.apache.org
7. https://github.com/chorsystem/middleware



(a) Absolute size

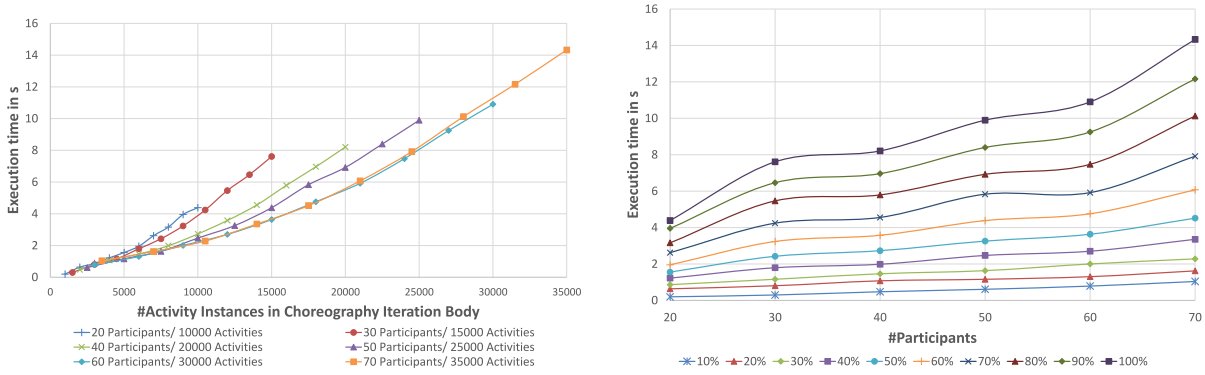


(b) Relative size

Fig. 7. Execution time (in seconds) for choreography models with constant number of participants and varying size of activities and message links.

algorithm that are too small for meaningful measurements (in the range of milliseconds). Each model increases linearly in size in terms of included activities (from 10K to 35K) and message links between the participants (from 1.5K to 5.25K). Subsequently, we generated for each choreography model 10 iteration bodies with varying wavefronts and increasing size of executed activity instances and traversed message link instances. Fig. 7 summarizes our findings in terms of execution time for the 6 generated choreography models. As can be seen from Fig. 7a, the execution time of the rewinding algorithm is quadratic with the number of generated activity and message link instances in the iteration body of each individual model. Likewise, the comparison of the execution time between different models for the same relative size of iteration body, e.g., 50 percent of included activity and message link instances, shows a quadratic increase of execution time (Fig. 7b). The quadratic growth of the execution time with the number of included activity and message link instances can be explained by looking at Algorithms 2 (traverseInstanceGraph) and 3 (handleSendingActivities). If *no loops* are present, Algorithm 2 traverses each participant instance graph exactly once ($\mathcal{O}(n)$, for all participants), while the *succ* function used in Algorithm 3 also traverses the complete participant instance in the worst case. This results in $\mathcal{O}(n \times n)$ for each participant instance in the choreography model, as confirmed by Figs. 7a and 7b.

The second part of the performance evaluation consists of the random generation of 6 new choreography models having 500 activities per participant, but increasing linearly in size in terms of participants (20 to 70) and message links (1.5K to 5.25K). Again, 10 different iteration bodies were



(a) Absolute size (b) Relative size

Fig. 8. Execution time of choreography models with constant number of activities and varying size of participants and message links.

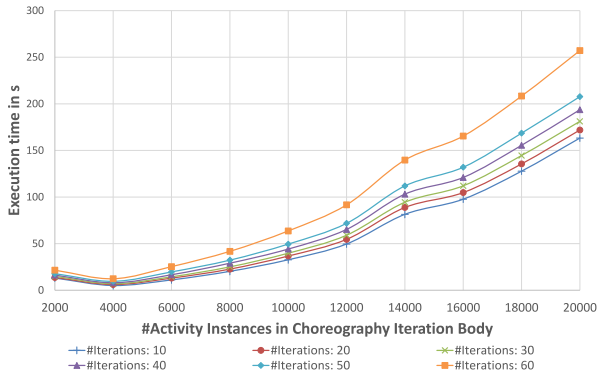
generated for each model using the same approach as described above. Fig. 8 shows the execution time for the 10 generated iteration bodies of all 6 choreography models. As before, the observed execution time growth is quadratic with the model size (Fig. 8a). However, the overall execution time is much smaller compared to the models with increasing number of activities per participant. This can be explained by the fact that the *succ* function (cf. Definition 14) has to traverse a much smaller amount of activity instances per participant, thus, yielding a faster execution time. This explanation is also supported by the execution times of the iteration bodies with the same relative size across the 6 generated models (Fig. 8b), which show a linear increase.

In the third part, in order to evaluate the impact of loops to our approach, we conducted three different measurements varying one of three parameters: number of loops per participant, number of activities per loop (loop size), and number of iterations per loop. For each of the three measurements, we generated a model having 10 participants with 2K activities each and 3K message links between them. Again, for each of the three measurements, 10 choreography iteration bodies of increasing size were generated. If the generated choreography wavefront lies inside a loop activity, we always simulate the worst case scenario where all specified iterations of the corresponding loop activity have been executed. Fig. 9 summarizes the findings of the conducted measurements. In general, all measurements confirm the quadratic increase of execution time. Fig. 9a shows execution times when varying the number of iterations per loop. This increase between the graphs is induced by the increasing number of activity instances that need to be traversed with each additional loop iteration. Fig. 9b shows that the loop activity itself does not influence the execution time when only increasing the loop size parameter per participant. Fig. 9c shows the measurements when only increasing the number of loops per participant. Again, the loop activities itself do not add any significant overhead to the execution time and its growth is also quadratic. We can therefore conclude that the presence of loops in the choreography models, while adding to the overall execution time due to an increased number of activity instances to be traversed, does not significantly affect the performance of our algorithm.

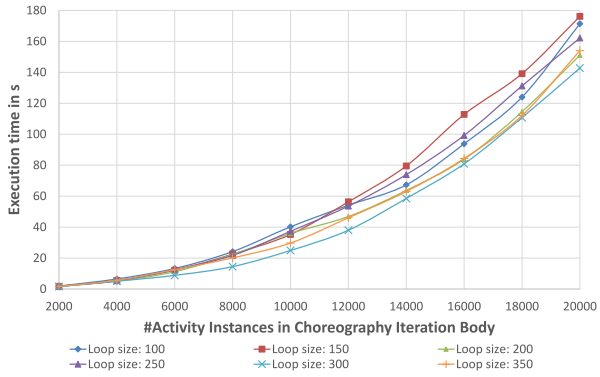
6 Related Work

There are several areas related to our work, such as ad hoc repetition in process instances, rollback-recovery and log-based protocols, and algorithms for consistent global state and predicate detection in distributed systems. In literature, the concept of ad hoc repetition in process instances is well studied. For example, in [21] concepts and algorithms for pre-modeled or ad hoc backward jumps, which enable the repeat of logic in process instances enacted by the ADEPT system are presented. The Kepler system supports the concept of smart re-runs [22] enabling scientists to repeat parts of a scientific workflow with a different set of parameters. Previously stored provenance information is used to avoid the repetition of parts of the workflow that do not change the overall outcome of the scientific experiment. In [23] process flexibility approaches are studied and classified by type. Our concept for rewinding and repeating choreography instances could be classified as *Flexibility by Deviation*—deviating from the specified control flow in the model. Similarly, our repetition is one form of the *Support for Instance-Specific Changes* as described in [24] for individual process instances. However, none of the above mentioned works consider choreography instances and the implications of messages sent between the participant instances.

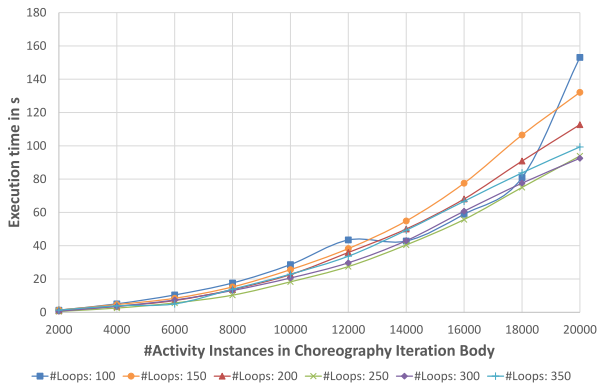
Our approach also bears some resemblance to rollback-recovery and log-based protocols facilitating distributed state restoration in message passing systems [25], [26]. These approaches provide means to reset a system of communicating processes to a rollback point in the execution of a program in case of failures. Failures may occur in any participating process and have an effect on other processes in the system due to passed messages. The proposed protocols then identify rollback points either by using log information or by sending checkpoint information. However, there are major differences between this family of protocols and our approach. First, we operate one a much higher level with regard to the employed languages. While the distributed protocols simply assume a set of communicating low level processes of some program execution, our approach operates on complex choreography and workflow instances. They are explicitly described by corresponding language constructs. These include the support for parallel execution inside one participant. Second, the intent and scopes of the approaches are quite different. The rollback-recovery protocols are triggered automatically in case of failures. While our approach can be used to react to failures, it is rather meant as a means of user-driven control of a choreography instance during execution to flexibly react to events. Furthermore, we do not only support the restoration to a previous state using the re-execute operation but rather



(a) #Loops=100, Loop Size=3, Increasing #Iterations



(b) #Loops=1, #Iterations=10, Increasing Loop Size



(c) Loop Size=3, #Iterations=10, Increasing #Loops

Fig. 9. Execution time (in seconds) of choreography models with loops (20K Activities, 3K Message Links).

also enable the repeated execution of logic with the iterate operation resembling an enforced loop without it being explicitly modeled. Apart from the language level the mentioned arguments also apply for more recent approaches [27], [28], [29] for checkpointing and rollback-recovery for composite web services/workflows.

With regard to robustness and reliability of workflow executions, [30] proposes a pattern-based approach for specifying transactional properties of service compositions. Furthermore, [31] provides an approach for mining logs of transactional workflows in order to improve the original model. While we ensure robustness by allowing the user to actively influence the execution of a choreography instance as necessary for explorative modeling and execution in domains such as eScience, these approaches focus more on the reliability by design. However, a combination of both approaches seems promising.

Another class of algorithms possessing similarities to our approach are algorithms for observing consistent global states and predicate detection. For example, Chandy and Lamport [32] introduce an algorithm for determining a snapshot of global state in a distributed system in order to detect stable predicates such as termination. However, finding a rewinding point in a choreography instance would not be possible with this kind of algorithm as it can be seen as a unstable predicate detection problem. Marzullo and Neiger [33] present one of the first algorithms using a centralized monitor to record state changes of all processes in a distributed system in order to evaluate if certain unstable predicates hold during execution. This is done by constructing state lattices. The problem of predicate detection in general is NP-complete. However, more efficient solutions for restricted scenarios exist [34]. In our approach, we are also able to achieve an efficient solution by choosing a graph-based data structure fitting our purposes.

7 Conclusions and Future Work

In this article, we motivated the need for the capability to repeat partially or completely the logic in a choreography instance with a clear focus on the eScience community. Toward this goal, we presented a formal model describing choreography models and instances while also considering loops and multiple instances of a particular participant. Based on the formal model, we introduced the concept of repeating logic in choreography instances, which we also expressed through the steps of a corresponding method. We distinguish between iteration, which executes logic again without undoing already completed work, and re-execution, which aims at the compensation of already completed work before executing it again. We defined an algorithm that is able to automatically identify the rewinding points for each involved participant instance. Furthermore, we presented a system for execution and monitoring of choreography instances that supports the proposed concepts and method. The rewinding algorithm has been experimentally evaluated in terms of performance and shown to be acceptable for the purposes of the application domain.

In future, we plan to evaluate our approach and the supporting system in cooperation with other groups of scientists in the context of the SimTech project.[8] In addition, we will work towards enabling transparent data provisioning among participants in flexible choreographies in a manner decoupled from the actual choreography conversations.

Acknowledgments

This work is funded by the project FP7 EU-FET 600792 ALLOW Ensembles and the German DFG within the Cluster of Excellence (EXC310/2) SimTech. Furthermore, the authors would like to thank David R. Schäfer for the fruitful discussions during the writing of this article.

References

[1] T. Hey, S. Tansley, and K. Tolle Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Cambridge, U.K.: Microsoft Research, 2009.

8. http://www.simtech.uni-stuttgart.de

[2] R. Barga and D. Gannon, "Scientific versus business workflows," in *Workflows for e-Science*. Berlin, Germany: Springer, 2007, pp. 9–16.
[3] M. Sonntag and D. Karastoyanova, "Model-as-you-go: An approach for an advanced infrastructure for scientific workflows," *Grid Comput.*, vol. 11, no. 3, pp. 553–583, 2013.
[4] M. Sonntag and D. Karastoyanova, "Ad hoc iteration and re-execution of activities in workflows," *Int. J. Adv. Softw.*, vol. 5, no. 1/2, pp. 91–109, 2012.
[5] A. Weiß and D. Karastoyanova, "Enabling coupled multi-scale, multi-field experiments through choreographies of data-driven scientific simulations," *Comput.*, vol. 98, no. 4, pp. 439–467, 2016.
[6] A. Weiß, V. Andrikopoulos, M. Hahn, and D. Karastoyanova, "Rewinding and repeating scientific choreographies," in *Proc. Confederated Int. Conf. Move Meaningful Internet Syst.*, 2015, pp. 337–347.
[7] G. Decker, O. Kopp, and A. Barros, "An introduction to service choreographies," *Inf. Technol.*, vol. 50, no. 2, pp. 122–127, 2008.
[8] D. Molnar, et al., "Multiscale simulations on the coarsening of Cu-rich precipitates in a-Fe using kinetic Monte Carlo, molecular dynamics and phase-field simulations," *Acta Materialia*, vol. 60, no. 20, pp. 6961–6971, 2012.
[9] W. van der Aalst and M. Weske, "The P2P approach to interorganizational workflows," in *Proc. 13th Int. Conf. Adv. Inf. Syst. Eng.*, 2001, pp. 140–156.
[10] F. Leymann and D. Roller, *Production Workflow - Concepts and Techniques*. Englewood Cliffs, NJ, USA: PTR Prentice Hall, 2000.
[11] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.
[12] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for modeling choreographies," in *Proc. IEEE Int. Conf. Web Services*, 2007, pp. 296–303.
[13] A. Weiß and D. Karastoyanova, "A life cycle for coupled multi-scale, multi-field experiments realized through choreographies," in *Proc. IEEE 18th Int. Enterprise Distrib. Object Comput. Conf.*, 2014, pp. 234–241.
[14] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Reading, MA, USA: Addison-Wesley, 2004.
[15] P. Newcomer and E. Bernstein, *Principles of Transaction Processing*, 2nd ed. San Mateo, CA, USA: Morgan Kaufmann, 2009.
[16] P. A. Berstein, V. Hadzilacos, and N. Goodman, *Concurreny Control and Recovery in DataData Systems*. Reading, MA, USA: Addison-Wesley, 1987.
[17] A. Weiß, V. Andrikopoulos, S. Gómez Sáez, M. Hahn, and D. Karastoyanova, "ChorSystem: A message-based system for the life cycle management of choreographies," in *Proc. Confederated Int. Conf. Move Meaningful Internet Syst.*, 2016, pp. 503–521.
[18] G. Decker, O. Kopp, F. Leymann, and M. Weske, "Interacting services: From specification to execution," *Data Knowl. Eng.*, vol. 68, no. 10, pp. 946–972, 2009.
[19] OASIS, "Web services business process execution language version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
[20] A. Weiß, Chorsystem project website. (2017). [Online]. Available: http://www.iaas.uni-stuttgart.de/chorsystem/
[21] M. Reichert, P. Dadam, and T. Bauer, "Dealing with forward and backward jumps in workflow management systems," *Softw. Syst. Model.*, vol. 2, no. 1, pp. 37–58, 2003.
[22] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the Kepler scientific workflow system," in *Provenance and Annotation of Data*. Berlin, Germany: Springer, 2006, pp. 118–132.
[23] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *Advances in Enterprise Engineering I*. Berlin, Germany: Springer, 2008, pp. 16–30.
[24] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.
[25] D. L. Russell, "State restoration in systems of communicating processes," *IEEE Softw. Eng.*, vol. SE-6, no. 2, pp. 183–194, Mar. 1980.
[26] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surveys*, vol. 34, no. 3, pp. 375–408, 2002.
[27] H. Mansour and T. Dillon, "Dependability and rollback recovery for composite Web services," *IEEE Trans. Services Comput.*, vol. 4, no. 4, pp. 328–339, Oct.–Dec. 2011.
[28] S. D. Urban, L. Gao, R. Shrestha, and A. Courter, "Achieving recovery in service composition with assurance points and integration rules," in *Proc. OTM Confederated Int. Conf. On Move Meaningful Internet Syst.*, 2010, pp. 428–437.
[29] Y. Xiao and S. Urban, "Using rules and data dependencies for the recovery of concurrent processes in a service-oriented environment," *IEEE Trans. Services Comput.*, vol. 5, no. 1, pp. 59–71, Jan.–Mar. 2012.
[30] S. Bhiri, W. Gaaloul, C. Godart, O. Perrin, M. Zaremba, and W. Derguech, "Ensuring customised transactional reliability of composite services," *J. Database Manage.*, vol. 22, no. 2, pp. 64–92, 2011.
[31] W. Gaaloul, K. Gaaloul, S. Bhiri, A. Haller, and M. Hauswirth, "Log-based transactional workflow mining," *Distrib. Parallel Databases*, vol. 25, no. 3, pp. 193–240, 2009.
[32] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985.
[33] K. Marzullo and G. Neiger, "Detection of global state predicates," in *Proc. Int. Workshop Distrib. Algorithms*, 1991, pp. 254–272.
[34] C. M. Chase and V. K. Garg, "Detection of global predicates: Techniques and their limitations," *Distrib. Comput.*, vol. 11, no. 4, pp. 191–201, 1998.

Andreas Weiß received the MSc degree in business information systems from the University of Stuttgart and the University of Hohenheim, Stuttgart, Germany, in 2013. Since then, he is working as a research associate in the Institute of Architecture of Application Systems (IAAS), University of Stuttgart. His main topics of research interest lay in the flexible modeling and execution of choreographies for scientific use cases such as simulations.

Vasilios Andrikopoulos received the PhD degree from Tilburg University, the Netherlands. He is an assistant professor of software engineering with the University of Groningen. His research is in the area of software architectures for service-oriented, cloud-based, and hybrid systems and infrastructures, as well as software engineering with an emphasis on evolution and adaptation. He has experience in research and teaching Database Systems and Management, Software Modeling and Programming, Business Process Management and Integration, and Service Engineering. He has participated in a number of EU projects, including the Network of Excellence S-Cube.

Michael Hahn received the Dipl-Inf degree in computer science from the University of Stuttgart, in 2013. Currently, he works as a research associate and PhD degree at the Institute of Architecture of Application Systems (IAAS), University of Stuttgart. His research interests include service oriented computing and architectures, focusing on the optimization of data exchange between choreographed services based on use cases from the e-Science domain.

Dimka Karastoyanova received the PhD degree in computer science from Technische Universität Darmstadt, Germany. She is an associate professor of data science and business intelligence with the Kühne Logistics University. Her research and teaching activities are in the fields of service-oriented computing, BPM and flexible workflow management, data science, and cloud computing, and in application domains like eScience, supply chain management and logistics. She was on the research and management team of European Projects like the NoE S-Cube, IP SUPER and FET ALLOWEnsambles, and of the German Cluster of Excellence SimTech.

Modeling Latent Relation to Boost Things Categorization Service

Yuanyi Chen, Jingyu Zhang, Liting Xu, Minyi Guo, *Senior Member, IEEE*, and Jiannong Cao, *Fellow, IEEE*

Abstract—While it is well understood that the Internet of things (IoT) offers the capability of integrating the physical world and the cyber world, it also presents many significant challenges with numerous heterogeneous things connected and interacted, such as how to efficiently annotate things with semantic labels (i.e., things categorization) for searching and recommendation. Traditional ways for things categorization are not effective due to several characteristics (e.g., thing's text profiles are usually short and noise, things are heterogeneous in terms of functionality and attributes) of IoT. In this paper, we develop a novel things categorization technique to automatically predict semantic labels for a given thing. Our proposed approach formulates things categorization as a multi-label classification problem and learns a binary support vector machine classifier for each label to support multi-label classification. We extract two types of features to train classification model: 1) explicit feature from thing's profiles and spatial-temporal pattern; 2) implicit feature from thing's latent relation strength. We utilize a latent variable model to uncover thing's latent relation strength from their interaction behaviours. We conduct a comprehensive experimental study based on three real datasets, and the results show fusing thing's latent relation strength can significantly boost things categorization.

Index Terms—Things categorization, multi-label classification, interaction behaviours, latent variable model, internet of things

1 Introduction

Recent years have witnessed numerous physical things (e.g., mobile phones, wallets and key-chains) embedded with sensing, communication and computing capabilities are being inter-connected to form an Internet of things, which is mainly attributed to the rapid advances in identification technologies and micro sensors, such as radio frequency identification (RFID), self-powered sensors, and nano technology sensors. Physical things embedded with smart sensors are seamlessly integrated into the information network, people can query and change their state and associated information over the Internet. Interconnection of physical things providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications, such as supply chain management, smart healthcare and intelligent transportation. Meanwhile, the IoT also presents a few significant challenges with increasing heterogeneous things participate in sensing and communicating, such as how to efficiently annotate these heterogeneous things with semantic labels for browsing, searching and recommendation. Traditional ways (for a review see Section 2) for things categorization will suffer serious challenges due to unique characteristics in IoT:

- Text-based categorization methods [33], [36], [37] cannot achieve satisfactory performance as the text profiles of things are usually short and noise in IoT. Additionally, labels are usually expensive and unlabelled things are abundant in IoT.
- Semantic-based categorization methods [1], [6], [7], [8], [23] are not effective as they require time-consuming preparation of prior knowledge, such as manually defining the descriptions of things and their corresponding concepts under a uniform format like Resource Description Framework (RDF).
- Link-based categorization methods [19], [22], [30] are infeasible in IoT since the connection of things in IoT are usually implicit, unlike people has observable links in social network or web-pages are linked by universal resource locator (URL) in Internet.

Fortunately, the interaction behaviours of things can be easily recorded and obtained using ubiquitous sensing technologies, such as RFID and sensor readings. These interaction behaviours, which embedded with rich spatial-temporal information and implicitly imply the regularities of users, provide us a new approach to uncover the latent connection of things. Things are discrete without explicit connection in IoT, but human and things will interact in daily activities, and these interactions can provide rich information (e.g., activity, location and time) for uncovering thing's implicit connection. Our proposed approach can derive the latent relation strength among things from their interaction behaviour and further form a relation graph of things, where their implicit connections can be revealed. This kind of relation analysis can boost many valuable services in IoT, such as:

- *Y. Chen is with the Department of Computer Science and Technology, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: cyyxz@sjtu.edu.cn.*
- *J. Zhang, L. Xu, and M. Guo are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: zhangzhang@sjtu.edu.cn, {xu-lt, guo-my}@cs.sjtu.edu.cn.*
- *J. Cao is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China. E-mail: csjcao@comp.polyu.edu.hk.*

Manuscript received 28 Nov. 2016; revised 7 May 2017; accepted 5 June 2017. Date of publication 14 June 2017; date of current version 13 Oct. 2020. (Corresponding author: Yuanyi Chen.) Digital Object Identifier no. 10.1109/TSC.2017.2715159

Things clustering, which aims to cluster heterogeneous things into different groups according to a predefined proximity measure. The key of things clustering is designing proximity metric to measure the similarity of things. However, traditional ways based on text features or thing's attributes are not effective as the unique characteristics in IoT (e.g., the text descriptions of things are usually incompleted, things are heterogeneous in term of different attributes thus cannot be represented in a uniform space).

The relation analysis of things can enhance the performance of things clustering in terms of things tend to interact with other things with similar characteristics according to [40]. Therefore, things clustering can be solved by some graph-based algorithms (e.g., community detection algorithm [38]) based on thing's relation graph.

Things Categorization, which aims to automatically predict the labels for a given thing. Millions of things connected and interacted in the IoT will result in serious challenges for things management and network scaling, while effectively things categorization is an essential step to cope with these challenges. For instance, in education scenario, things categorization enables learners to get rich data and further improve their knowledge [7]. As mentioned earlier, text-based models [33], [37] or link-based models [22], [30] are not effective for things categorization in the IoT.

Fortunately, things interactions are not completely random as human daily activities usually follow a regular pattern [34], such as people usually cook in the kitchen and eat breakfast at 7:00-9:00 am. Therefore, things relation analysis based on their interactions can boost things categorization since different things used by the same person at the same location or time may be similar (e.g., having the same label).

Context-aware Activity Recognition, which aims to recognize human activities (e.g., eating, cooking and toileting) from sensor readings. Existing approaches usually use the sensor values of things as input to train a probabilistic model to find the most likely sequence of activities (e.g., Hidden Markov Model [32] and support vector machine [25]). However, these methods are inefficient when human activities are performed in a complex situation (i.e., interleaved or concurrent) [12].

Complex human activities can be defined as a task that several things interact at specific location at a certain timestamp (see in Table 2), and then can be modeled as a 3-dimensional tensor: $Activity \in R^{Things \times Location \times Time}$. After modeling the sensor readings with 3-dimensional tensor by deriving thing's latent relation strength from their interaction behaviours, activity recognition can be solved by finding a matching scheme to measure the similarity of two tensors.

In this paper, we present a novel things categorization technique for the IoT to automatically predict the labels for a given thing. We formulate things categorization as a multi-label classification problem and learn a binary SVM classifier for each label in the label space to support multi-label classification. We extract two types of features to train classification model: 1) explicit feature from thing's profiles and spatial-temporal pattern; 2) implicit feature from similar things in terms of thing's latent relation strength. The principle underlying our approach for modeling thing's latent relation strength is the homophily theory in thing's interaction [39], [40], which suggests the stronger the relation the higher likelihood that a certain type of interaction will take place between a pair of things. In this way, we consider that the latent relation strength directly impacts the interaction frequency of a pair of things, and further model the latent relation strength as a hidden cause of their interaction frequency.

The remainder of the paper is organized as follows: Section 2 surveys related work about things categorization in IoT. Section 3 describes the proposed approach for modelling the latent relation strength of things in detail. Section 4 demonstrates how to utilize the learnt latent relation strength to boost things categorization. Section 5 reports and discusses the experimental results. Finally, we present our conclusion and future work in Section 6.

2 Related Work

In this section, we survey related works about things categorization and discuss how these works differ from our study.

Text-Based Categorization. IoT things usually have short and noisy text descriptions, such as thing's name, manufacturer and instruction manual. Thus text-based methods can be used to label things, which first extract text-based features (e.g., term frequency and information gain) and then perform categorization with classifiers. To overcome some limitations of traditional text features, the work [33] proposed a novel feature selection method based on term frequency and T-test for text categorization, and [37] utilized the compactness of the appearances of the word and the position of the first appearance of the word to construct distributional features for text categorization.

Since assigning labels to large samples is costly and time-consuming, the work [36] proposed a web-assisted text categorization framework, which first extracted important keywords from the available labelled documents to form the queries, then utilized search engines to retrieve relevant documents for semi-supervised categorization. Unfortunately, this approach is impractical for things categorization in IoT as there is few information about physical things in Internet nowadays.

Semantic-Based Categorization. A few studies ([1], [6], [7], [8], [23]) have been proposed to label things using Semantic Web technologies. The idea behind semantic-based categorization is that first define a metadata model to describe all the cyber-physical characteristics (e.g., geophysical, functional and non-functional) of a thing, then use ontology language description logic to label physical things in terms of different dimensions (e.g., spatial, temporal and thematic).

To enrich the description of thing's characteristics for educational purpose, the work [7] exploited shared vocabularies for categorization by three steps: 1) defining a data model for representing Point-of-Interest; 2) mapping the relational database to the data model; 3) generating RDF data and enriching with links to related data. The work [8] proposed an IoT semantic categorization framework, which representing the model data as linked data and associating with the existing data on the Web (e.g., Linked Open Data). The work [6] proposed a hierarchical context model based on ontology to label things and their contextual relationships. The work [23] further utilized Time-of-Arrival for thing's geospatial categorization in IoT. The drawbacks of semantic-based categorization methods include: 1) The time-consuming preparation of prior knowledge, such as

defining the descriptions of things and their corresponding concepts under a uniform format like RDF; 2) Most of semantic-based categorization methods are based on problem-solving principle, which defines the ontology related to a certain task (e.g., home energy [1] and education purpose [7]) or activities thus are lack of scalability.

Link-Based Categorization. IoT things are implicitly connected in a network or graph by some attributes (e.g., location, owner and manufacturer), thus link-based methods [19], [22], [30], [31] can leverage these connections to improve categorization performance. For example, the work [35] labelled things by modelling things as web tables with headers and cell values. More exactly, this categorization process includes three steps: 1) querying the background knowledge base sources to generate initial ranked lists of candidate assignments for schemas, content values and relations between schemas; 2) using a probabilistic graphical model to capture the correlation between schemas, content values and schema relations to make class, entity and relation assignments; 3) producing linked data triples after the mapping is complete and performing things categorization using link-based methods [19].

More recently, increasing studies [2], [4], [17] aimed at giving social-like capabilities to the things in IoT, namely social internet of things (SIoT) based on the notion of social relationships among things. In SIoT [2], things are able to interact with other things in an autonomous way with respect to the owners, and can easily crawl the IoT made of billions of things to discover services and information in a trust-oriented way. For example, SIoT [2] described four kinds of relationships for things in IoT: co-location relationship, co-work relationship, co-owner and social relationship. Lilliput [4] further extended the SIoT by integrating things as well as online social networks, which is an ontology-based platform by fusing online social networks and things as a social graph. Socialite [17] utilized semantic model for the SIoT, which includes device types, capabilities, users, relationships and rules leveraging such models. Therefore, many link-based methods [22], [30] can be utilized to label things by modeling heterogeneous things and their relationships with a graph. Unfortunately, link-based methods are ineffective for things categorization in IoT due to 1) acquiring a sufficient number of labelled things to enable accurate learning for link-based categorization usually are expensive or impractical; 2) SIoT may ignore some implicit factors that may influence categorization performance (such as usefulness and availability), for instance, Microwave and Toaster may have different manufacture or owner, but both they are kitchen appliances and can heat foods.

To our best knowledge, only several studies [39], [40] focus on boosting things categorization by exploring regularities in the interactions between human and things. These approaches discovered the latent relation strength of things by mining three dimensional information of the interaction behaviours: user, temporarily and spatiality. However, we find these approaches mentioned above fail to model thing's latent relation strength and their interaction behaviours by analyzing three real datasets, for example, we observe that the interaction probability of two things and their history interaction frequency follows a roughly power law distribution. Additionally, these studies infer thing's relation strength without considering their attributes profiles (e.g., the manufacturer, type and capability).

Our proposed approach differs from the above-mentioned works in the following three aspects: 1) we regard things categorization as a multi-label classification problem and learn a binary SVM classifier for each label in the label space to perform things categorization; 2) we extract two types of features to train SVM classifier, one is explicit features from thing's text profiles and spatial-temporal pattern, another is implicit features from thing's latent relation strength; 3) we derive thing's latent relation strength by jointly considering thing's profile similarities and interaction behaviours with a latent variable model. Recently, latent variable model has been widely used in a few studies on text mining. For instance, latent semantic analysis (LSA) [18] supposed that there is an underlying semantic structure in text and the relationship between terms and documents can be derived in this semantic space. Several studies [20], [42] based on LSA are proposed to deal with short text classification. Probabilistic latent semantic analysis (pLSA) [14] extended LSA by explicitly defining latent topic of a document as the latent variable during a random process, which is widely used in text summarization [26] and image annotation [43]. Latent Dirichlet Allocation (LDA) [3] further extended pLSA by adding priors (Dirichlet Distribution) to the document collection, which occupies an important position in many fields of text mining (such as text classification [5] and review-based sentiment analysis [24]).

3 Modeling the Latent Relation of Things from Interaction Behaviour

In this section, we first present the problem statement of modeling thing's latent relation strength from their interaction behaviours. Then detail the proposed approach, a latent variable model to derive thing's latent relation strength.

3.1 Problem Statement

For ease of the following presentation, we first define the key data structures and notations used in the proposed approach. Table 1 lists the relevant notations used in this paper.

Definition 1 (Thing). *A Thing o_i in IoT, denote by $< ID_i, A_i >$, where ID_i is the identifier of o_i and $A_i = \{a_i^1, \ldots, a_i^j, \ldots, a_i^{|A_i|}\}$ is the attributes set of o_i (e.g., type, color and manufacturer). a_i^j is the value of the jth attribute of o_i.*

As identified by [13] and [16], things in IoT are sensing and actuating physical devices that providing the ability to share information across platforms through a unified framework. Thus things has the following three characteristics: 1) physical devices. Thing's attributes are directly related to the physical characteristics of devices [9]; 2) embedded-in sensors, which are utilized to provide sensing, computing and communication ability; 3) unique identity. For example, the associated IP address can be utilized as thing's identifier. Things concept can be explained by the following example.

Example 1. Considering a thing o named smart oven[1] (as shown in Fig. 1), which is a physical oven but embedded

1. https://www.wired.com/2016/03/tovalas-smart-oven-wants-replace-microwave/

TABLE 1
Notations Used in the Paper

SYMBOL	DESCRIPTION
O, T, Loc, U	the set of things, timestamps, locations, labels
N, Q, F, H	the number of things, timestamps, locations, interactions
A_i	the attribute set of thing o_i
a_i^j	the value of the jth attribute of o_i
$Y^{(ij)}$	the interaction set between thing o_i and o_j
$X^{(ij)}$	the variables to capture the tendency of interactions
$z^{(ij)}$	A similarity vector based on thing's attributes
$I^{(ij)}$	the latent relation strength between o_i and o_j
Ω_i	the k-neighbour set of o_i in terms of relation strength
w	A K-dimensional similarity vector to be estimated
σ^2	the variance of Gaussian distribution
$\alpha_l, \beta_l, \theta_l$	the parameters of power law distribution
$G = \{V, E, W\}$	thing's top-k relation graph, $V = V_s \cup V_r$
V_s, V_r	the labelled things set and unlabelled things set
M	the transition matrix of random walk with restart
F_{Latent}	implicit features from thing's relation graph
$F_{Cluster}$	implicit features by clustering thing's interactions
F_{text}	text-based features from thing's text descriptions
F_S	spatial features from thing's spatial pattern
F_T	temporal features from thing's temporal pattern

in several sensors (e.g., laser scanner for reading bar codes on the lids of compatible meals, temperature and humidity sensor for detecting whether the heat and humidity inside the oven is optimal, and built-in WiFi for downloading new and newly perfected recipes from the cloud). For an oven, its physical characteristics consist of $< type : appliance >, < color : black >, < manufacturer : Tovala >, < function : heatingfood >$, thus we can obtain its attributes set: $A_o = \{appliance, black, Tovala, heatingfood\}$. According to thing's definition, we denote thing o as $< 192.168.1.125, A_o >$ by using IP address as thing's identifier.

Definition 2 (Interaction Behaviour). *Interaction behaviour among things happens when people use things in daily activities (e.g., Preparing breakfast, Dish washing and Brushing teeth). Let* $O = \{o_1, o_2, \ldots, o_N\}, T = \{ts_1, ts_2, \ldots, ts_Q\}$ *and* $Loc = \{loc_1, \ldots, loc_F\}$ *denote the set of things, timestamps and locations, respectively. An interaction between* o_i *and* o_j*, denote by* $y \in Y^{(ij)} = \{y_1^{(ij)}, y_2^{(ij)}, \ldots, y_H^{(ij)}\} = \{< o_i, o_j, ts, loc > | o_i, o_j \in O \bigwedge ts \in T \bigwedge loc \in Loc\}$*, indicates that a user used* o_i *and* o_j *in location* loc *at timestamp* ts*. To extract the timestamp of thing's interaction behaviours, we divide a day into 24 hourly slots. To this end, we generate the total number of hashed time slots is 24, denote as* $TS=\{ts_1, ts_2, \ldots, ts_{24}\}$*. For instance, if two things interact at 1:32 pm, 3/15/2016, the time slot of this interaction is* ts_{14}.

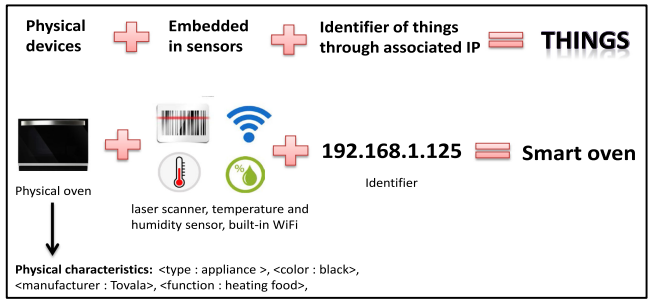


Fig. 1. An illustrative example of thing and its attribute set.

TABLE 2
An Example of Things Interaction Behaviour

Daily activity	Preparing breakfast,3/5/2016, 08:13:12, 08:24:18
Things	Freezer, Microwave, Sink faucet - hot, Plate, Pan
Starting Time	08:15:38, 08:17:21, 08:19:35, 08:22:14, 08:20:25
Ending Time	08:21:24, 08:23:19, 08:20:11, 08:23:38, 08:20:46

In our experiment, we utilize a context-aware experience sampling tool (CEST) and state-change sensors to collect thing's interaction behaviours, i.e., the state-change sensors recorded data about the movement of things and the participants used CEST to record information about their activities. During the study, each participant was given a PDA to run CEST tool. The participant utilizes the CEST to select the activity what he/she is doing, and records the start and end time of this activity. Fig. 2 shows an example of the type of data that was collected by the state-change sensors and CEST. As shown in Fig. 2a, the starting and ending time of things are automatically recording by their embedded-in state-change sensors. Then, we can obtain the participated things of an activity by observing the sensor activations during the activity duration (as shown in Fig. 2b). As shown in Table 2, we generate the daily activity (Preparing breakfast) involved five things, and consider

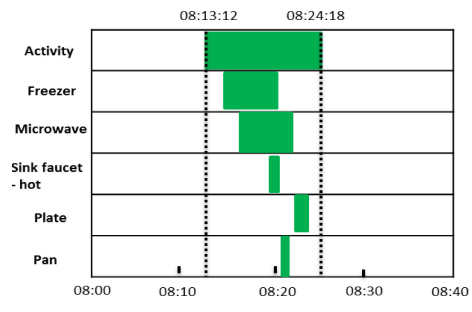
Activity Information from CEST Tool

Activity	Day	Starting Time	Ending Time	Location
Preparing breakfast	3/5/2016	08:13:12	08:24:18	Kitchen

Sensor Activations Information from Built-in Sensors

Sensor ID	Day	Starting Time	Ending Time	Location	Thing
36	3/5/2016	08:15:38	08:21:24	Kitchen	Freezer
26	3/5/2016	08:17:21	08:23:19	Kitchen	Microwave
28	3/5/2016	08:19:35	08:20:11	Kitchen	Sink faucet - hot
131	3/5/2016	08:22:14	08:23:38	Kitchen	Plate
19	3/5/2016	08:20:25	08:20:46	Kitchen	Pan
	.....	.....			

(a)



(b)

Fig. 2. An illustrative example of collecting thing's interaction behaviours.

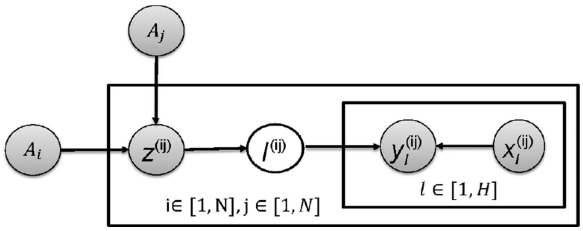


Fig. 3. Graphical model of learning thing's latent relation strength.

there is an interaction behaviour between each pair of things during this activity.

Definition 3 (Latent Relation Strength). *The latent relation strength between o_i and o_j denote by $I^{(ij)}$,which is determined by i) the attribute similarity of o_i and o_j; and ii) the interaction behaviours between o_i and o_j. In other words, for the larger latent relation strength $I^{(ij)}$, two things (o_i and o_j) are required to be more similar in term of either attributes or are more likely to interact with each other.*

With the aforementioned definitions, the problem of modeling thing's latent relation strength can be formally stated as follows:

Given a set of things: $O = \{< ID_1, A_1 >, < ID_2, A_2 >, \ldots, < ID_N, A_N >\}$; and their history interaction behaviours $Y = \{y_1, y_2, \ldots, y_H\}$, the problem of modeling latent relation strength aims to discover the implicit connection of things by exploiting their attributes and observable interaction behaviours.

3.2 Approach

In this section, we first describe the modeling part of the proposed approach, a latent variable model to infer thing's latent relation strength from their interaction behaviours, and then present its inference process.

3.2.1 Model Description

Previous studies [39], [40] have shown that homophily is ubiquitous in IoT, which suggests the likelihood that a certain type of interaction will take place between a pair of things relate positively to their latent relation strength. In this way, we model thing's latent relation strength as the hidden cause for their interaction behaviours. Such interactions could be, for example, preparing breakfast, eating lunch and brushing teeth. We further consider thing's latent relation strength as a hidden effect of thing's profile similarities. The profiles similarity are caused by thing's attributes, such as the manufacturer, the functionality and the geographic locations that they belong to, etc.

Formally, let $Y^{(ij)} = \{y_1^{(ij)}, y_2^{(ij)}, \ldots, y_H^{(ij)}\}$ denote the interaction behaviours between o_i and o_j, $I^{(ij)}$ denote the latent relation strength between o_i and o_j. Then, we utilize a graphical model to represent the influence caused by the profiles similarity to $I^{(ij)}$, as well as the influence of $I^{(ij)}$ on interaction behaviours, as shown in Fig. 3. In this figure, the gray-colored nodes depict observed variables (i.e., $z^{(ij)}, Y^{(ij)}$ and $\{x_1^{(ij)}, x_2^{(ij)}, \ldots, x_H^{(ij)}\}$), which are all visible in the training phase.

The detailed description of variables in this figure is explained as follows:

- $z^{(ij)}$ denotes the profiles similarity of o_i, o_j, which is a K-dimensional vector calculated based on the attributes set of o_i and o_j (i.e., A_i and A_j).
- $I^{(ij)}$ is the latent relation strength between o_i and o_j, which is a hidden factor for thing's interaction behaviours and influenced by their profiles similarity.
- $X^{(ij)} = \{x_1^{(ij)}, x_2^{(ij)}, \ldots, x_H^{(ij)}\}$ are auxiliary variables that we introduce to increase the accuracy of the model. Such variables capture auxiliary causes of the interactions which are independent of the latent relation strength. For example, the total number of interactions that a thing participated in represents its intrinsic tendency to interact, which can moderate the effect of latent relation strength on interaction behaviours.

Our model represents the relationships among these variables by modeling the conditional dependencies (as shown in Fig. 3), so the joint distribution decomposes as follows:

$$P\left(I^{(ij)}, Y^{(ij)} | A_i, A_j) = P(I^{(ij)} | A_i, A_j) \prod_{l=1}^{H} P(y_l^{(ij)} | I^{(ij)}, x_l^{(ij)}\right). \tag{1}$$

Given the attributes of o_i and o_j, we model the conditional probabilities $P(I^{(ij)}|A_i, A_j)$ using the widely-used Gaussian distribution

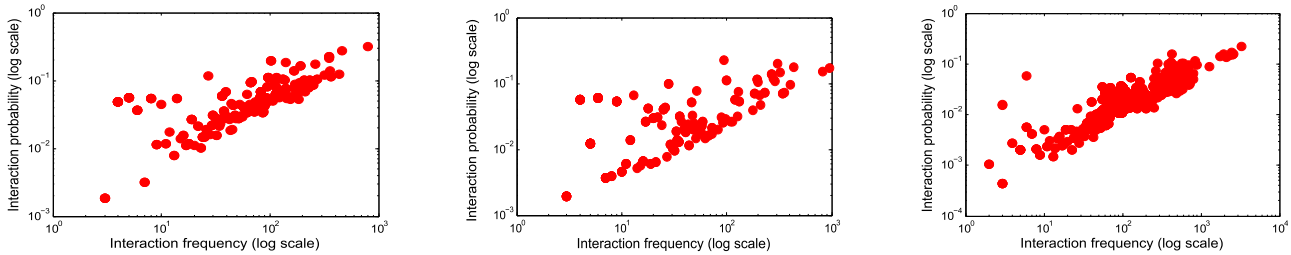
$$P(I^{(ij)} | A_i, A_j) = (w^T z^{(ij)}, \sigma^2), \tag{2}$$

where w is a $K-$dimensional weight vector to be estimated and σ^2 is the variance of Gaussian model, $z^{(ij)}$ is the profiles similarity based on A_i and A_j.

For modeling the dependency between $Y^{(ij)}$ and $I^{(ij)}, X^{(ij)}$, we analyze the characteristics of thing's interaction behaviours using three real world datasets: 1) Our dataset is collected by 13 participants during six months, which consists of 32,716 interaction records from 196 things ; 2) MIT S1, S2. The two datasets are published by the AI group in MIT [28], which consists of 503 interaction records from 146 things in total. More details of these datasets are shown in Table 3. Fig. 4 shows the likelihood of two things may interact as a function of their historical interaction frequency. As shown in Fig. 4, we observe there exists a positive correlation between the likelihood of two things may interact and their historical interaction frequency, indicating a clustering phenomenon in thing's interaction behaviours. This phenomenon may be intuitively explained by the following tendencies: 1) things with similar attributes (e.g., provided similar services and located in the same geographic location) tend to interact; 2) things with the same label (e.g., cooking tools and office supplies) tend to interact. As mentioned earlier, we consider thing's latent relation strength is determined by their attribute similarity and

TABLE 3
Detailed Information of the Three Datasets

	MIT S1	MIT S2	Dataset 3
# of things	76	70	196
# of daily activities	33	35	93
# of interaction records	295	208	32,716
# of participants	1	1	13
# of collecting period (days)	14	14	180



(a) MIT S1 (b) MIT S2 (c) Our dataset

Fig. 4. Fraction of interaction probability as a function of thing's historical interaction frequency.

interaction behaviours We believe that this clustering phenomenon in thing's interaction behaviours can be exploited for uncovering thing's latent relation strength. Thus, in the following, we study and model thing's latent relation strength and their interaction frequency.

Based on Fig. 4, we intuitively think the distribution follows a roughly power-law form. Even though the right part of the figure increases linearly (i.e., increases exponentially in regular scale) and thus fits power-law distribution very well, the left part may sometimes deviate irregularly (i.e., the probability is high at some points). A reasonable explanation is that the likelihood of two things may interact cannot judge from few interactions. Generally speaking, the fact that two things with more historical interactions tend to interact is confirmed in our data analysis. Moreover, the linear portion of the plot in Fig. 4 covers the majority (90 percent) of the interaction behaviours In this way, we model the dependency between $Y^{(ij)}$ and $I^{(ij)}$, $X^{(ij)}$ with a power-law distribution

$$P(y_l^{(ij)}|I^{(ij)}, x_l^{(ij)}) = (\alpha_l I^{(ij)} + \beta_l x_l^{(ij)})^{\theta_l}, \tag{3}$$

where α_l, β_l and θ_l are parameters of power law distribution to be estimated, $l = 1, 2, \ldots, H$.

We further add L_2 regularizes on these hyper parameters (e.g., α_l, β_l, θ_l) to avoid over-fitting, which can be regarded as Gaussian prior

$$\begin{aligned} &P(\alpha_l, \beta_l) \propto e^{-(\lambda_1/2)(\alpha_l^2+\beta_l^2)}, l = 1, \ldots, H \\ &P(\theta_l) \propto e^{-(\lambda_2/2)(\theta_l)^2}, l = 1, \ldots, H \\ &P(w) \propto e^{-(\lambda_3/2)(w^T w)}. \end{aligned} \tag{4}$$

The dataset are represented as a set of thing pairs: $\Phi = O \times O$, denoted as $D = \{(i_1, j_1), \ldots, (i_N, j_N)\}$. During training phase, the variables $z^{(ij)}, y_l^{(ij)}$ and $x_l^{(ij)}$ are all visible, $(i, j) \subseteq \Phi$. According to Equation (1), given all the observed variables, the joint probability is shown as

$$\begin{aligned} &\prod_{l=1}^{H} P(\Phi|w, \alpha_l, \beta_l, \theta_l) P(w, \alpha_l, \beta_l, \theta_l) = \\ &\prod_{(i,j)\in D} P(I^{(ij)}|z^{(ij)}, w) P(w) \prod_{l=1}^{H} P(D|I^{(ij)}, x_l^{(ij)}, \alpha_l, \beta_l, \theta_l) P(\alpha_l, \beta_l, \theta_l) \\ &\propto \prod_{(i,j)\in D} \left(e^{-(1/2\delta^2)(w^T z^{(ij)} - I^{(ij)})^2} \prod_{l=1}^{H} (\alpha_l I^{(ij)} + \beta_l x_l^{(ij)})^{\theta_l} \right) \\ &e^{-(\lambda_3/2) w^T w} \prod_{l=1}^{H} e^{-(\lambda_2/2)(\theta_l)^2} e^{-(\lambda_1/2)(\alpha_l^2+\beta_l^2)}. \end{aligned} \tag{5}$$

3.2.2 Model Inference

We estimate the unknown model parameters $\Sigma = \{w, \alpha_l, \beta_l, \theta_l\}$ by maximizing the likelihood function as shown in Equation (5). As for the hyper parameters $\sigma^2, \lambda_1, \lambda_2, \lambda_3$, for simplicity, we take a fixed value ($\sigma^2 = 0.5$, $\lambda_1 = \lambda_2 = \lambda_3 = 0.01$) in experiment. Applying a logarithmic transformation to both sides of Equation (5), we obtain the following expression

$$\begin{aligned} L((i, j) \in D, w, \alpha_l, \beta_l, \theta_l) = &\sum_{(i,j)\in D} -\frac{1}{2\sigma^2}(w^T z^{(ij)} - I^{(ij)})^2 \\ &+ \sum_{(i,j)\in D} \sum_{l=1}^{H} \theta_l log(\alpha_l I^{(ij)} + \beta_l x_l^{(ij)}) - \frac{\lambda_3}{2}(w^T w) \\ &- \sum_{l=1}^{H} \frac{\lambda_2}{2} \theta_l^2 - \sum_{l=1}^{H} \frac{\lambda_1}{2}(\alpha_l^2 + \beta_l^2). \end{aligned} \tag{6}$$

Note the function L (see in Equation (6)) is concave, then we optimize the parameters $\alpha_l, \beta_l, \theta_l$ and variable $I^{(ij)}$ with a stochastic gradient descent algorithm. We use Netwton-Raphson algorithm to update these parameters in each iteration

$$I^{(ij)new} = I^{(ij)old} - \frac{\partial L}{\partial I^{(ij)}} \Big/ \frac{\partial^2 L}{\partial (I^{(ij)})^2} \tag{7}$$

$$\alpha_l^{new} = \alpha_l^{old} - \frac{\partial L}{\partial \alpha_l} \Big/ \frac{\partial^2 L}{\partial (\alpha_l)^2} \tag{8}$$

$$\beta_l^{new} = \beta_l^{old} - \frac{\partial L}{\partial \beta_l} \Big/ \frac{\partial^2 L}{\partial (\beta_l)^2} \tag{9}$$

$$\theta_l^{new} = \theta_l^{old} - \frac{\partial L}{\partial \theta_l} \Big/ \frac{\partial^2 L}{\partial (\theta_l)^2}. \tag{10}$$

Where the coordinate-wise gradients and the second order derivatives can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2017.2715159.

As for w, the coordinate-wise gradient is as following:

$$\frac{\partial L}{\partial w} = -\frac{1}{\sigma^2} \sum_{(i,j)\in D} z^{(ij)}(w^T z^{(ij)} - I^{(ij)}) - \lambda_3 w. \tag{11}$$

The root of $\partial L / \partial w = 0$ can be solved by ridge regression [11]

$$w = (\lambda_3 \sigma^2 I + Z^T Z)^{-1} Z^T C, \tag{12}$$

where $Z = [z^{(i_1 j_1)}, z^{(i_2 j_2)}, \ldots, z^{(i_N j_N)}]^T, C = [I^{(i_1 j_1)}, I^{(i_2 j_2)}, \ldots, I^{(i_N j_N)}]^T$, I is the identity matrix.

Algorithm 1 shows the procedure for optimizing these parameters, we optimize model parameters $\Sigma = \{w, \alpha_l, \beta_l, \theta_l\}$ using Newton-Raphson until converged.

Algorithm 1. The Algorithm for Optimizing Parameters

Require: Data samples $D = \{(i_1, j_1), \ldots, (i_N, j_N)\}$.
Ensure: Model parameters $\Sigma = \{w, \alpha_l, \beta_l, \theta_l | l = 1, 2, \ldots, H\}$.
1: **while** not converged **do**
2: **for** each Newton-Raphson step **do**
3: ***Step1: Estimate latent relation strength***
4: **for** $(i, j) \in D$ **do**
5: Update $I^{(ij)}$ according to Equation (7).
6: **end for**
7: ***Step2: Estimate parameters $\alpha_l, \beta_l, \theta_l$***
8: **for** $l = 1, 2, \ldots, H$ **do**
9: Update $\alpha_l, \beta_l, \theta_l$ according to Equations (8), (9), (10)
10: **end for**
11: **end for**
12: Update w according to Equation (12).
13: **endwhile**
14: **return** $\Sigma = \{w, \alpha_l, \beta_l, \theta_l | l = 1, 2, \ldots, H\}$.

4 Boosting Things Categorization Using Latent Relation

Modelling thing's latent relation strength can facilitate a few valuable services (e.g., things categorization, recommendation and searching) in IoT. Due to space constraints, we briefly introduce an important application: things categorization, which aims to automatically predict appropriate semantic labels that a given thing. A thing may be associated with multiple labels in IoT. For example, a microwave associated with a label *cooking* may also be tagged with *appliance*, and a television may label with *entertainment* and *appliance*. Therefore, things categorization can be formulated as a multi-label classification problem. In this study, we propose a method for things categorization by learning a binary SVM classifier for each label to support multi-label classification. To train the classification model, we extract two kinds of features for each thing: 1) implicit features from similar things in terms of the learnt latent relation strength, which is derived by building a top-k relation graph where similar things are linked by virtual edges; 2) explicit features of things, such as text features (e.g., Term Frequency or Term Frequency Inverse Document Frequency) and spatial-temporal pattern.

4.1 Implicit Features from the Learnt Latent Relation Strength

We extract the implicit features among things in order to formulate descriptive features of a given thing from its similar things. To capture the implicit features from similar things, we first construct a top-k relation graph of things based on the learnt relation strength. In thing's relation graph, things are linked by their latent relation strength, which is derived from their interaction behaviours. Then, we perform random walk with restart (RWR) [10] to derive the relation strength between each pair of things. The goal of RWR is to predict the label probability for a given thing by exploring the latent relation strength with similar things, and using the label probability as implicit feature for classification.

(1) Construct Top-k Relation Graph of Things (RGT). The idea of extracting implicit features is to infer descriptive features of a given thing from its neighbour and labelled things, since only few things are labelled in IoT. In this way, we construct a top-k relation graph of things by connecting things together.

Formally, let $G = < V, E, W >$ denote the top-k relation graph among things, where $V = V_s \cup V_r$ (V_s is the labelled vertex set and V_r is the unlabelled vertex set) is the set of nodes, E is the set of edges in G. For $o_i, o_j \in V_s$, we define $W_{ij} = 1$ if o_i and o_j have the same class label, 0 otherwise. If at least one of o_i, o_j is unlabelled, W_{ij} is defined as

$$W_{ij} = \begin{cases} exp(-1/\eta I^{(ij)}), & \text{if } o_j \in \Omega_i \text{ or } o_i \in \Omega_j \\ 0, & \text{otherwise} \end{cases}, \quad (13)$$

where Ω_i denotes the k nearest neighbour set of o_i in term of relation strength and Ω_j is also similar, η is a weight coefficient.

(2) Perform RWR on RGT. Let $\Omega_i = \{o_{j1}, o_{j2}, \ldots, o_{jk}\}$ be the set of k nearest neighbours which is connected with o_i in RGT, τ_i denotes a tag μ associated with thing o_i ($\mu \in U$). Let $P(\tau_i = \mu | \Omega_i)(\mu \in U)$ denote the probability that o_i may associate with label μ, we initialize $P(\tau_i = \mu | \Omega_i) = 1/|U|$ for unlabelled things, and $P(\tau_i = \mu | \Omega_i) = 1$ for labelled things if thing o_i has label μ or 0 otherwise. Then, we utilized RWR to find $P(\tau_i = \mu | \Omega_i)(\mu \in U)$ for each thing.

Without lost of generality, we assume the random walker starts from an unlabelled things o_i on graph G. Then, the random walker iteratively transmits to other nodes which have edges with o_i, with the probability that is proportional to the edge weight between them. At each step, o_i also has a restarting probability λ to return itself. We can obtain the steady-state probability of o_i by visiting other vertexes until the RWR process is converged. The RWR process can be formulated as

$$P_{t+1}(\tau_i = \mu | \Omega_i) = (1 - \lambda) M P_t(\Omega_i) + \lambda P_t(\tau_i = \mu | \Omega_i), \quad (14)$$

where $P_t(\tau_i = \mu | \Omega_i)$ represents the estimation probability in step t, $P_t(\Omega_i)$ denote the estimation probabilities of all nearest neighbours of o_i at step t, denote by $P_t(\Omega_i) = [P_t(\tau_{j1} = \mu | \Omega_{j1}), P_t(\tau_{j2} = \mu | \Omega_{j2}), \ldots, P_t(\tau_{jk} = \mu | \Omega_{jk})]$. M is the transition matrix, which is obtained based on weight matrix W by row normalization, as shown in

$$M = W \Psi^{-1}, \quad (15)$$

where $W = [w(i, j1), w(i, j2), \ldots, w(i, jk)]^T$, Ψ is the usual normalizer and defined as $\Psi = \sum_{js \in \Omega_i} W(i, js)$.

The label probability estimation for each label on a thing o_i can be obtained when the RWR process is converged, which are regarded as implicit features for SVM training (F_{Latent}). The process on how to extract implicit features can be explained by the following example.

Example 2. As shown in Fig. 5, suppose we have 5 things (three labelled things:{A,B,C} and two unlabelled things: {D,E}) and 4 labels (l1,l2,l3 and l4). After constructing top-2 relation graph of things based on their latent relation strength, we extract thing's implicit features by performing RWR on the relation strength according to Equation (14)

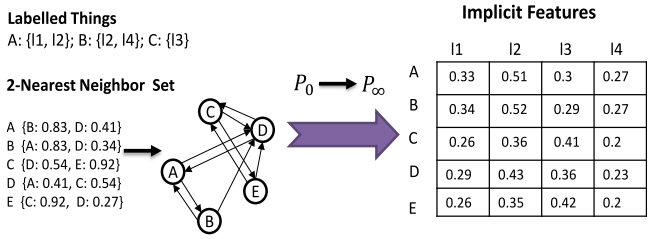
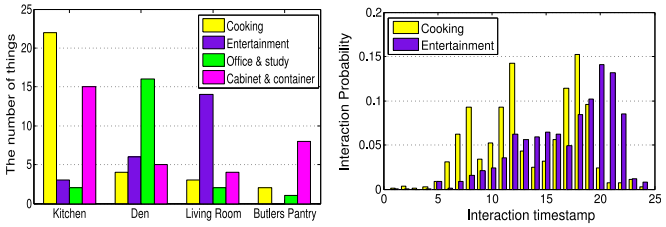


Fig. 5. An example of extracting implicit features from top-k relation graph.



(a) Spatial pattern (b) Temporal pattern

Fig. 6. Spatial-temporal pattern of things associated with different labels.

until converged. Finally, we obtain the label probability estimation for each possible label on a thing and regard the label probability estimation as implicit features. For instance, the implicit features of thing A are a 4-dimensional vector $[0.33, 0.51, 0.3, 0.27]$, while the implicit features of thing D is: $[0.29, 0.43, 0.36, 0.23]$.

Let $|U|$ denotes the number of labels, $|V|$ denotes the total number of things, and $|E|$ denotes the total number of edges in the relation graph. It takes $O(|U| * |V|)$ time to initialize the label probabilities for all things. Then at each iteration, we need to process each edge twice to update the label probabilities, once for each thing at each end of the edge. We also need $O(|U| * |E|)$ time to learn from the initial label probabilities, so the time complexity of each iteration is $O(|U| * (|E| + |V|))$. Therefore, the total time complexity for extracting implicit features is $O(t * |U| * (|E| + |V|))$, where t is the maximum number of iteration needed to reach the steady state. We will experimentally demonstrate that this algorithm converges in a few iterations. And since the number of labels $|U|$ is constant, the computational complexity is generally linear in the number of edges and nodes in the relation graph.

4.2 Explicit Features from Text and Spatial-Temporal Pattern

Things usually have some short and noisy text profiles (e.g., thing's color, type and manufacturer), which can be used to extract text-based features for multi-label classification. On the other hand, thing's interactions imply some spatial-temporal patterns as human daily activities usually follow a regular temporal pattern [34], for instance, people usually eat dinner at 5:00-7:00 pm, which means the interacted things tend to be *cooking* tools during the time slot. We extract three explicit features from thing's text profiles and spatial-temporal patterns for training classification model.

4.2.1 Text-Based Feature

We utilize the well-known Term Frequency Inverse Document Frequency (TF/IDF) to extract keywords from things' text descriptions [29], and the weight of keywords are regarded as text-based features (F_{text}).

4.2.2 Spatial Feature

To show the spatial pattern of thing's interaction behaviours in IoT, we aggregate the number of things associated with different labels at a specific location using a real-world dataset that consists of 196 things, more details of this dataset are shown in Table 3. As shown in Fig. 6a, things with different labels have very different spatial pattern corresponding to different locations. For instance, things associated with *Cooking* and *Cabinet & container* are mainly located in *Kitchen*, while things associated with *Office & study* are mainly located in *Den*. Therefore, we consider the spatial pattern to be a very useful feature for things categorization. Formally, we define spatial pattern as a F-dimensional vector: $F_S(i) = [SF^i(loc_1), SF^i(loc_2), \ldots, SF^i(loc_F)]$. Note that, $SF^i(loc_k)$ is computed as

$$SF^i(loc_k) = \frac{N^i(loc_k)}{\sum_{j=1}^{F} N^i(loc_j)}, \tag{16}$$

where $N^i(loc_k)$ is the number of interactions involved thing o_i in location loc_k, $\sum_{j=1}^{F} N^i(loc_j)$ is the total number of interactions involved thing o_i, and F is the number of locations.

4.2.3 Temporal Feature

Fig. 6b reports the hourly distribution of things associated with different labels at different timestamps using a real-world dataset collected over half years (see in Table 3). From this figure, we can observe two very different temporal patterns corresponding to two kinds of labels (i.e., *Cooking* and *Entertainment*). For instance, the interaction of things associated with label *Cooking* have clearly three peak periods, corresponding to breakfast, lunch and dinner time, respectively. On the contrary, for things associated with label *Entertainment*, the interaction has one peak period (from 6:00 to 10:00 pm). Therefore, the temporal pattern of thing's interaction is discriminative feature for distinguishing different labels, such as *Cooking* and *Entertainment*. Formally, we define temporal pattern as a T-dimensional vector: $F_T(i) = [TF^i(t_1), TF^i(t_2), \ldots, TF^i(t_T)]$. $TF^i(t_k)$ is computed as

$$TF^i(t_k) = \frac{N^i(t_k)}{\sum_{j=1}^{T} N^i(t_j)}, \tag{17}$$

where $N^i(t_k)$ is the number of interactions involved thing o_i at timestamp t_k, $\sum_{j=1}^{T} N^i(t_j)$ is the total number of interactions involved thing o_i, and T is the number of timestamps.

4.3 Things Categorization by Fusing Explicit Features and Implicit Features

In our approach, we first formulate things categorization as a multi-label classification problem and then decompose into several distinct single-label binary classification problems. For instance, as shown in Fig. 7, *Microwave* labelled *Cooking* is positive sample for a classifier for *Cooking* but negative sample for a classifier for *Storage*, while *Garbage*

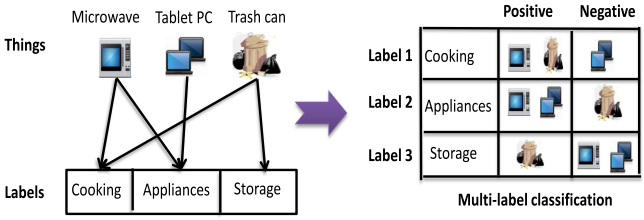


Fig. 7. An example of formulating things categorization to a multi-label classification problem.

Ambient Sensors	Activities	Tagged things
Force sensor or Pressure sensor	Sleeping, sitting, napping	Beds, couches
Contact sensors	Opening and closing of the doors, cupboards	Door frames, shower cupboards
Proximity sensors	Detecting close distance objects	Chairs, closets and taps
Infrared Receiver Sensor	Watching TV	Near the TV

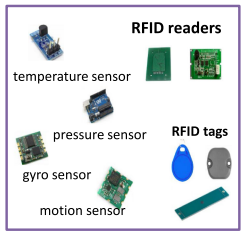


Fig. 8. Part of sensors and devices used in our experiment.

bag labelled *Storage* is positive sample for a classifier for *Storage* but negative sample for a classifier for *House Appliances*. To perform things categorization, our approach outputs the aggregation of the labels positively predicted by all the independent binary classifiers.

After extracting four kinds of features: 1) Implicit features (F_{Latent}) from thing's relation graph; 2) Text-based features (F_{text}) from thing's text descriptions; 3) Spatial features (F_S) from thing's spatial pattern when interacting; 4) Temporal features(F_T) from thing's temporal pattern when interacting, we combine the features ($F_{Latent} + F_{text} + F_S + F_T$) together as inputs for training a set of binary SVM classifier for things categorization.

5 Experiment Evaluation

In this section, we first describe the experiment settings including data sets, baseline methods and evaluation metric. Then, we report and discuss the experimental results.

5.1 Datasets

Three real world datasets about things interaction in IoT: two public datasets from MIT [28] AI group and one collected dataset (Dataset 3) in our experiment, are used for experimental evaluation. More details of these datasets are reported in Table 3.

- *MIT Dataset*. The first two datasets (MIT S1 and MIT S2) are published by the AI group in MIT, which collected two subject's daily activities during two weeks. The first subject was a 30-year-old woman who spent free time at home, and the second was an 80-year-old woman who spent most of her time at home. Both subjects lived alone in one-bedroom apartments, 77 and 84 sensors are installed in everyday things (e.g., *Microwave, Refrigerators and Stoves*) of the two subject's apartment, respectively. Each data collection board was marked on a plan-view of the environment for collecting data after these sensors were installed, and the location (e.g., *Bathroom*) and type (e.g., *Toaster*) of each thing associated sensor was prior known. For recording the subject's activities information, a context-aware experience sampling tool was used for labelling activities. Finally, 76 and 70 things that have participated in interactions are used to conduct experiments.
- *Our Dataset*. Our experiment environment includes one workspace (e.g., office, laboratory and meeting room) and two smart houses (e.g., bedroom, living room and kitchen). In our experiment, there are 196 things are tagged with RFID and various sensors (e.g., motion, pressure and temperature sensors, as shown in Fig. 8) for collecting interaction behaviours. For generating thing's interaction behaviours, three types of information need to be recorded: 1) Activity information. To obtain the activity information, each participant utilized a context-aware experience sampling tool to mark and record their activities when interacting with things; 2) Temporal information. To map the interacting time to the corresponding timestamps, we split one day into 24 timestamps with one hour as an interval as mentioned earlier; 2) Spatial information. For static things (e.g., cabinet, toaster and door), the spatial information is prior known. For mobile things (e.g., RFID-tagged remote control and coffee cup), we utilized a fingerprint-based positioning algorithm to estimate the unknown location [44]. Thirteen participants participated in the data collection phase during six months, and more than 32,000 interaction behaviours of things are recorded in the experiment.

We manually labelled these things with different semantic labels as the ground-truth data for performance evaluation. Note that some things are labelled with multiple labels (e.g., *Microwave* is labelled with both *Cooking* and *House Appliances*, *Television* is labelled with both *Entertainment* and *House Appliances*), thus a thing may belong to multiple categories. Finally, we manually labelled these things with 798 different labels, more details information can be found in Appendix B, available in the online supplemental material, due to space limitation.

5.2 Baseline Methods

We extract five kinds of feature for a thing to train a set of binary SVM classifiers (the whole set of features are shown in Table 4), and aggregate of the labels positively predicted by all the independent binary classifiers as things categorization result. Among the five features, $F_{Cluster}$ has not been discussed before, which means deriving thing's implicit features by clustering thing's interactions directly instead of using the proposed graphical modeling approach. The feature extraction process of $F_{Cluster}$ is similar to F_{Latent}, which first constructs top-k relation graph of things based on

TABLE 4
Features for Things Categorization

Features	Description
F_{text}	The text-based feature using TF/IDF (Section 4.2.1)
F_S	The spatial pattern of interaction behaviours(Section 4.2.2)
F_T	The temporal pattern of interaction behaviours(Section 4.2.3)
$F_{Cluster}$	The label probabilities for labels by clustering interactions
F_{Latent}	The label probabilities for labels on a thing (Section 4.1)

TABLE 5
An Example of Text Description for MIT Datasets

Activity	Description
Preparing a snack	Domestic work,Preparing a snack
Doing laundry	Domestic work,Clean house
Bathing	Personal needs,Personal hygiene

thing's interactions and then performs RWR on the relation graph to extract thing's implicit features.

Based on these features, we evaluated 8 methods for things categorization as listed in Table 6. Among the 8 methods, TE needs to extract TF/IDF features from thing's text profiles. Thing's text profiles have not been discussed before, we detail it in the following. For MIT datasets, we utilize the activity description that things participated in as thing's text profiles to extract text-based features. For instance, a *Closet* participate in three kinds of activities: {*Preparing a snack,Doing laundry,Bathing*}, then we combine the descriptions of the three activities (as shown in Table 5) as its text description, i.e., "Domestic work,Preparing a snack, Clean house, Personal needs, Personal hygiene". For things of our collected dataset, we utilize the text description from E-commerce company (e.g., ebay[2] and Taobao[3]) as thing's text profiles to extract text-based features.

5.3 Evaluation Metrics

We use two widely used metrics (Hamming Loss and F-measure) for multi-label classification to evaluate the performance, which are defined as:

- *Hamming Loss*, which is used to evaluate how many times a thing is misclassified, i.e., a label not associated with a thing is predicted or a label associated with the thing is not predicted. Formally, the Hamming Loss is defined as

$$HammingLoss = \frac{1}{|D_{te}|}\sum_{i\in D_{te}}\frac{HD(v_i,\overline{v_i})}{|L|}, \tag{18}$$

  where $|D_{te}|$ is the number of test samples, $|L|$ is the number of labels, v_i and $\overline{v_i}$ are the ground truth and prediction vectors for testing thing o_i. $HD(v_i,\overline{v_i})$ is the hamming distance between v_i and $\overline{v_i}$.
- *F-measure*, which is a particular kind of average between precision and recall that has been widely used in many prediction problems including binary classification, multi-label classification and structured output prediction. Let v_i and $\overline{v_i}$ denote the ground truth and prediction vectors for testing thing o_i, the F-measure is defined as

$$F-measure = \frac{1}{|D_{te}|}\sum_{i\in D_{te}}\frac{2\times|v_i\bigcap\overline{v_i}|}{v_i+\overline{v_i}}. \tag{19}$$

Due to the small size of dataset, we perform five-fold cross-validation and also report the corresponding standard deviation as error bar for each case. First, each dataset was randomly split into 5 equal groups (N = 5). Second, trains the model on 4 groups of data, and records the error for the excluded data. This process is repeated 10 times, each time records the performance (Hamming Loss and F-measure) for the excluded data set. Finally, this whole procedure is repeated 10 times with different random splits of the data to produce the final results. We report the mean of the performance and standard deviation produced with the 50 (5x10) sets of test data as the ultimate experiment results.

2. http://www.ebay.com/
3. http://www.taobao.com/

TABLE 6
Methods for Comparison

Method	Description
TE	Using F_{text} to train SVM classifier
S	Using F_S to train SVM classifier
T	Using F_T to train SVM classifier
EF	Combination of F_{text}, F_S and F_T to train SVM classifier
CL	Using $F_{Cluster}$ to train SVM classifier
IF	Using F_{Latent} to train SVM classifier
CL+EF	Combination of F_{text}, F_S, F_T and $F_{Cluster}$ to train SVM classifier
IF+EF	Combination of F_{text}, F_S, F_T and F_{Latent} to train SVM classifier

5.4 Parameter Setting

For our dataset, we utilize five attributes to capture the profiles similarity of each pair of things (o_i, o_j), which is defined as: $z^{(ij)} = [z_1^{(ij)}, \ldots, z_5^{(ij)}]^T$ and the meaning of the five features are reported in Table 7. For MIT datasets, we utilize two attributes to capture the profiles similarity for each pair of things (o_i, o_j): $z^{(ij)} = [z_2^{(ij)}, z_3^{(ij)}]^T$, since the other three types of information are not provided.

5.5 Experiment Results

We conduct two groups of experiments and report their results. The first group is to perform parameter turning for models (IF and CL) using implicit features. The second group is to compare the effectiveness and efficiency of models using explicit features and implicit features for things categorization, respectively.

5.5.1 Impact of Model Parameters

Tuning algorithm parameters, such as the parameter λ of RWR process and the number of neighbours (k) for constructing top-k relation graph, are critical to the performance of methods (IF and CL) using implicit feature. We tune λ and k on the three datasets, and plot the Hamming Loss and F-measure with different values in Fig. 9.

Set the number of neighbours for constructing relation graph equals to 10, we test the performance of IF by varying λ, and present the results in Figs. 9a and 9b. As mentioned

TABLE 7
The Profile Similarity Features of Things

Feature	Description
$z_1^{(ij)}$	1 if o_i and o_j have the same manufacturer, 0 otherwise
$z_2^{(ij)}$	1 if o_i and o_j are owned by the same user, 0 otherwise
$z_3^{(ij)}$	1 if o_i and o_j are located in the same place, 0 otherwise
$z_4^{(ij)}$	1 if o_i and o_j have the same functionality, 0 otherwise
$z_5^{(ij)}$	1 if o_i and o_j have the same color, 0 otherwise

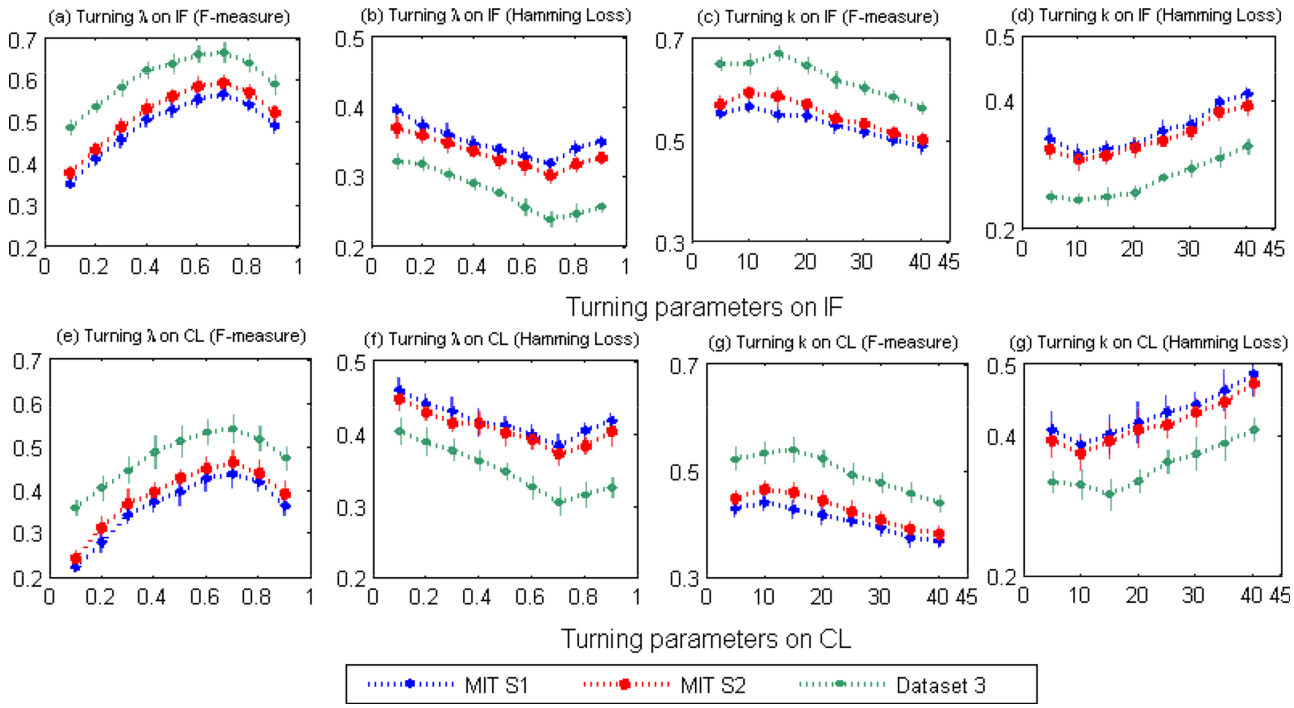


Fig. 9. Impact of parameters (λ and k) for IF and CL (mean plus standard error bars).

earlier, for each dataset, we perform 5-fold cross-validation and repeat 10 times for each cross-validation, and report the corresponding standard deviation as error bar for each case. From the figure, it is observed that best Hamming Loss and F-measure are reached when $\lambda = 0.7$. We further observe both the Hamming Loss and F-measure slightly increase with the increasing of λ from 0.1 to 0.7, and then decrease when λ is greater than 0.7. The reason is that the convergence of RWR is determined by λ, i.e., the greater λ results in the faster convergence, and further bring better performance. But a larger λ will cause a high probability to back to the target thing, thus reducing the number of neighbours with high latent relation strength and further decreasing the performance. As shown in Figs. 9e and 9f, similar results are also observed in turning λ for CL (for example, the mean of F-measure and Hamming Loss achieve the best results (43.74 and 38.16 percent, respectively) when $\lambda = 0.7$ for MIT S1.).

Set $\lambda = 0.7$, Figs. 9c 9d report the performance of IF with different number of neighbours (k), where k is in the range [5,10,...,40], because there are 342 things in total and a greater value of k is usually ignored when constructing top-k relation graph. From the two Figures, we observe the best Hamming Loss and F-measure are reached when k = 10 and 15 on MIT datasets and Dataset 3, respectively. The is because that our dataset has much more things than MIT datasets. However, the performance decreases with increasing k, since a greater k will bring in some noisy neighbours thus may decrease the performance. Similar results are also observed in turning k for CL, for example, the mean of F-measure and Hamming Loss achieve the best results (50.88 and 30.52 percent, respectively) when $k = 15$ for Dataset 3.

5.5.2 Explicit Features versus Implicit Features

In this part, we compare the effectiveness and efficiency of models using explicit features and implicit features for things categorization. We evaluate the categorization effectiveness from two aspects: 1) the performance of explicit Features (TE, S, T and EF), implicit Features (CL and IF) and their hybrid (EF+CL and EF+IF) with fixed mark-off rate. Here the mark-off rate means the ratio of unlabeled things. In this case, we perform 5-fold cross-validation and repeat 10 times for each cross-validation, and report the mean and the corresponding standard deviation as error bar; 2) the performance of five methods (EF, CL, IF, EF+CL and EF+IF) with different mark-off rate. In this case, we randomly removed the category labels of a certain percentage (named testing things with mark-off rate) from each category of the ground-truth dataset. The methods are used to recover the category labels for those testing things. For each case, we report the average performance and corresponding standard deviation as error bars by repeating the experiments 10 times.

Performance Comparison. We compare the performance of 8 methods (TE, S, T, EF, CL, IF, EF+CL, EF+IF) on the three datasets, as shown in Table 8.

For methods (TE, S, T and EF) using explicit features, we can observe from Table 8 that: 1) For methods using explicit features, EF that combines the spatial, temporal and text information always outperforms the baseline methods (TE, S, T), which merely utilizes one type of feature. For instance, EF outperforms TE by 22.09 and 11.73 percent on MIT S1 and MIT S2 in terms of F-measure, respectively. This result suggests that, fusing spatial, temporal and text feature is beneficial for improving the performance of things categorization; 2) For Dataset 3, TE achieves much worse performance than S and T in terms of both Hamming Loss and F-measure. The reason is that Dataset 3 utilizes the description from E-commerce company as thing's text description to extract text-based features, which usually are short and noisy. For example, a text description from E-commerce site

TABLE 8
Performance (%) Comparison with Different Baselines (F1=F-measure, HL=Hamming Loss)

Method	MIT S1		MIT S2		Dataset 3	
	F1 (%)	HL(%)	F1 (%)	HL(%)	F1 (%)	HL(%)
TE	47.31 ± 2.84	39.42 ± 2.47	59.05 ± 2.38	45.71 ± 1.94	30.17 ± 2.15	43.31 ± 2.42
S	35.02 ± 2.71	46.31 ± 1.87	50.08 ± 2.62	40.17 ± 2.36	41.59 ± 3.12	37.94 ± 2.86
T	39.42 ± 2.67	40.15 ± 2.89	57.16 ± 3.12	36.31 ± 2.79	50.62 ± 3.23	36.83 ± 2.89
EF	69.38 ± 2.47	26.89 ± 2.06	70.78 ± 2.95	24.28 ± 1.89	66.14 ± 1.83	22.11 ± 2.37
CL	43.74 ± 3.45	38.16 ± 2.79	46.38 ± 3.77	37.15 ± 2.93	53.88 ± 3.25	30.52 ± 3.76
IF	56.43 ± 2.95	31.85 ± 1.85	59.22 ± 2.46	30.13 ± 2.18	64.37 ± 2.79	23.93 ± 2.12
EF+CL	72.69 ± 2.74	24.17 ± 2.18	73.17 ± 3.35	20.71 ± 2.76	73.58 ± 2.53	20.19 ± 2.49
EF+IF	77.15 ± 2.36	20.89 ± 2.14	79.92 ± 3.27	16.58 ± 2.17	83.51 ± 2.78	14.13 ± 2.86

for cabinet is "the home cabinet with simplicity of modern style, two / three door", then the keywords extracted from this text are {cabinet, door}. Therefore, the cabinet is likely to be misclassified as door. The results suggest that text-based features based on the well-known TF/IDF feature for things categorization are not effective in IoT, since the text descriptions of things are usually short and noisy.

For methods using implicit features (CL and IF), we observe from Table 8 that IF outperforms CL significantly in terms of both F-measure and Hamming Loss, showing the advantages of using graphical model to mine thing's latent relation strength and derive implicit features. For instance, the F-measure of IF is about 56.43 percent on MIT S1, 59.22 percent on MIT S2 and 64.37 percent on Dataset 3, the performance is improved by 12.69 percent (MIT S1), 12.84 percent (MIT S2) and 10.49 percent (Dataset 3) compare with CL respectively. The reasons for better precision are: 1) CL extracts implicit features from thing's relation graph by clustering thing's interactions directly, which are powerless to capture information from things without interactions for deriving categorization features. On the contrary, our proposed graphical model can be applied in two ways. First, if both things attributes set and their interaction behaviours are known, we can estimate the latent relation strength based on both things attributes set and their interaction behaviours. Second, when the interaction behaviours are unobserved, we can estimate thing's latent relation from their attributes similarity. This in fact demonstrates a strength of our proposed graphical model: the lower part of the model is generative so that the overall model will not suffer much from missing interaction behaviours during training. Once the model is learned, for new data the latent variables can be inferred using only the upper level of variables in the model; 2) our proposed graphical model introduces a few auxiliary variables that capture auxiliary causes of thing's interactions, which can moderate the effect of latent relation strength on interaction behaviours thus increase the accuracy of the model. CF intuitively clusters thing's interactions for predicting labels, particularly the betweenness centrality is strongly biased towards nodes with high degree, or nodes that are central in large local groups of nodes [15]. Moreover, the random walk may terminate with a high likelihood when reaches an unlabeled nodes with numerous interactions during extracting implicit features [27].

For methods using both explicit features and implicit features (EF+CL and EF+IF), we can observe from Table 8: 1) the methods using hybrid features outperform merely using explicit feature (EF) or implicit features (CL or IF) significantly in terms of both Hamming loss and F-measure. For example, EF+IF outperforms EF by around 7.97 percent on MIT S1, 9.14 percent on MIT S2, and around 17.37 percent on Dataset 3 in terms of F-measure. This result shows the unified method (EF+IF) is superior to the state-of-art method in terms of categorization effectiveness, which suggests that the learnt latent relation strength from thing's interactions behaviours can significantly boost things categorization; 2) the performance improvement of our dataset is much higher than the other two datasets for both EF+CL and EF+IF. This is because our dataset consists of much more interaction behaviours than MIT datasets, which can be utilized to learn thing's latent relation strength better.

Performance with Different Mark-Off Rates. We investigate the impact of different mark-off rates to the performance of EF, CL, IF, EF+CL and EF+IF. As shown in Fig. 10, the performance of all methods with different feature sets degrade to some extent as the mark-off rate increases. Nevertheless, EF+CL and EF+IF show better performance consistently than CL and IF over all mark-off rates as they include both explicit features and implicit features. For example, the F-measure of EF+IF on our dataset is 78.44 percent when the mark-off rate is 40 percent, while 54.02 percent of EF with the same mark-off rate. This clearly demonstrates the effectiveness of hybrid features by combining both implicit feature and explicit feature. We also observe the unified method (EF+CL and EF+IF) can achieve considerable performance even when the mark-off rates are relatively high, while explicit features perform poorly with few labeled things to train model. For example, the F-measure of IF+EF on MIT S2 drops 5.6 percent when the mark-off rate increases from 40 to 60 percent, while 18.3 percent of EF with the same condition. This is because explicit features require either enough text profiles or obvious spatial-temporal pattern to extract features for training model, while hybrid features can utilize implicit feature extracted from thing's relation graph to boost things categorization even with few labeled samples.

Efficiency of Extracting Implicit Features. The time complexity of the proposed things categorization model consists of three parts: 1) the first part is the graphical model for inferring the model parameters. Since this part can be done in offline phase, the learned parameter values can be applied to estimate the latent relation strength for a new pair of things in constant time; 2) the second part is

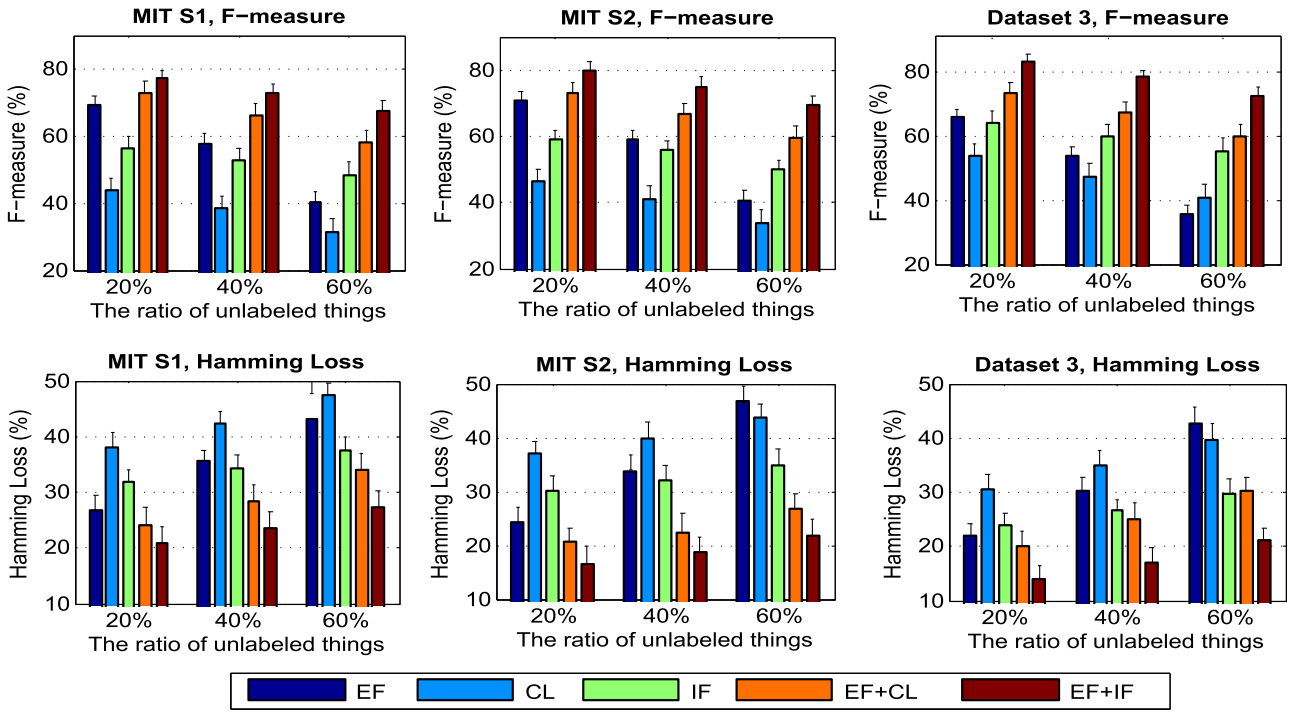


Fig. 10. Impact of mark-off rate on the three datasets.

extracting implicit features. We have proved in Section 4.1 that the time complexity of this part is generally linear in the number of edges and nodes in the relation graph. We will show the feature extraction process will converge fast in the following experiments; 3) the third part is multi-label classification by SVM model, which is scalable to big datasets as suggested by a few studies [21], [41]. Therefore, the proposed things categorization model is scalable to large dataset.

We use $\varepsilon = 10^{-5}$ as the termination condition of iterations. The iteration numbers when implicit feature extraction process terminates are plotted in Fig. 11a by varying the training sample sizes. From this figure, we observe the iteration numbers when implicit feature extraction process terminates are less than 400 for the three datasets, which shows the feature extraction process is scalability to large dataset. We further report the run time with different training sample sizes by setting the iteration number as 500 in Fig. 11b. From this figure, we can observe time complexity is generally linear in the ratio of unlabeled nodes as expected.

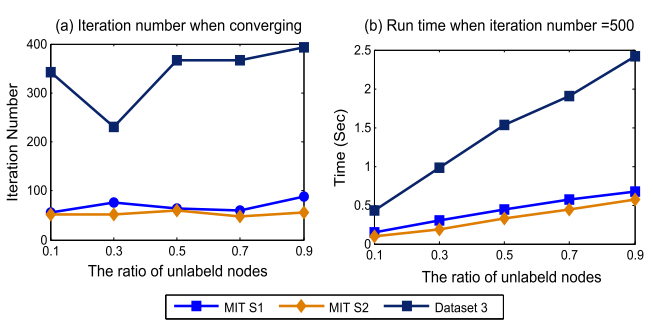


Fig. 11. The efficiency of extracting implicit features: (a) the comparison of iteration numbers when converging; (b) the comparison of run time.

6 Conclusion

In this paper, we investigate things categorization problem, which aims to automatically associate things with semantic tags in IoT. Things categorization is a crucial prerequisite for a few valuable services in IoT, such as things browsing, searching and recommendation. We propose a novel things categorization algorithm which learns a binary SVM classifier for each type of label. For training SVM classifier, we extract two kinds of features: explicit features and implicit features. More exactly, we extract three types of explicit features: text feature from thing's text profiles, spatial feature from thing's location distribution and temporal feature from the hourly distribution of thing's interaction. For extracting the implicit feature, we first construct a relation graph based on the learnt latent relation strength from thing's interaction behaviours, then exploit thing's relatedness to generate implicit feature. Finally, we conduct a comprehensive experimental study based on three real datasets. Experimental results show that this proposed approach significantly outperforms state-of-art methods based on explicit features, showing the superiority of our approach and also supporting the assumption that the latent relation strength among things can boost things categorization.

As future work, we plan to facilitate more valuable services in IoT based on the learnt latent relation strength of things from their interaction behaviours, such as, things searching, clustering and service discovery.

Acknowledgments

This work is sponsored by the National Basic Research 973 Program of China (No. 2015CB352403), the National Natural Science Foundation of China (NSFC) (61261160502, 61272099), the Program for National Natural Science

Foundation of China / Research Grants Council (NSFC/RGC)(612191030), the Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT), the Scientific Innovation Act of STCSM (13511504200), and EU FP7 CLIMBER project (PIRSES-GA-2012-318939).

REFERENCES

[1] I. Atanasov, A. Nikolov, E. Pencheva, R. Dimova, and M. Ivanov, "An approach to data annotation for internet of things," *Int. J. Inf. Technol. Web Eng.*, vol. 10, no. 4, pp. 1–19, 2015.

[2] L. Atzori, A. Iera, and G. Morabito, "From "smart objects" to "social objects": The next evolutionary step of the internet of things," *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 97–105, Jan. 2014.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003.

[4] J. Byun, S. H. Kim, and D. Kim, "Lilliput: Ontology-based platform for IoT social networks," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2014, pp. 139–146.

[5] X. Chen, Y. Xia, P. Jin, and J. A. Carroll, "Dataless text classification with descriptive LDA," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2224–2231.

[6] Y. Chen, J. Zhou, and M. Guo, "A context-aware search system for internet of things based on hierarchical context model," *Telecommun. Syst.*, vol. 62, no. 1, pp. 77–91, 2016.

[7] M. Coccoli and I. Torre, "Interaction with objects and objects annotation in the Semantic Web of things," in *Proc. Int. Conf. Database Manage. Syst.*, 2014, pp. 383–390.

[8] S. De, P. Barnaghi, M. Bauer, and S. Meissner, "Service modelling for the internet of things," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, 2011, pp. 949–955.

[9] F. C. Delicato, P. F. Pires, and T. Batista, *Resource Management for Internet of Things*. Berlin, Germany: Springer, 2017.

[10] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 3, pp. 355–369, Mar. 2007.

[11] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, Berlin, Germany: Springer, 2001.

[12] T. Gu, Z. Wu, X. Tao, H. K. Pung, and J. Lu, "epSICAR: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition," in *Proc. 7th Annu. IEEE Int. Conf. Pervasive Comput. Commun.*, 2009, pp. 1–9.

[13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[14] T. Hofmann, "Probabilistic latent semantic indexing," in *Proc. 22nd Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 1999, pp. 50–57.

[15] I. Hulpus, C. Hayes, M. Karnstedt, and D. Greene, "Unsupervised graph-based topic labelling using dbpedia," in *Proc. 6th ACM Int. Conf. Web Search Data Mining*, 2013, pp. 465–474.

[16] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the implementation of IoT for environmental condition monitoring in homes," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3846–3853, Oct. 2013.

[17] J. E. Kim, A. Maron, and D. Mosse, "Socialite: A flexible framework for social internet of things," in *Proc. 16th IEEE Int. Conf. Mobile Data Manage.*, 2015, pp. 94–103.

[18] T. K. Landauer, *Latent Semantic Analysis*. Hoboken, NJ, USA: Wiley, 2006.

[19] G. Limaye, S. Sarawagi, and S. Chakrabarti, "Annotating and searching web tables using entities, types and relationships," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 1338–1347, 2010.

[20] T. Liu, Z. Chen, B. Zhang, W.-Y. Ma, and G. Wu, "Improving text classification using local latent semantic indexing," in *Proc. 4th IEEE Int. Conf. Data Mining*, 2004, pp. 162–169.

[21] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, and W.-Y. Ma, "Support vector machines classification with a very large-scale taxonomy," *ACM SIGKDD Explorations Newslett.*, vol. 7, no. 1, pp. 36–43, 2005.

[22] L. K. McDowell, "Relational active learning for link-based classification," in *Proc. IEEE Int. Conf. Data Sci. Adv. Analytics*, 2015, pp. 1–10.

[23] I. Nevat, G. W. Peters, K. Avnit, F. Septier, and L. Clavier, "Location of things: Geospatial tagging for IoT using time-of-arrival," *IEEE Trans. Signal Inf. Process. Over Netw.*, vol. 2, no. 2, pp. 174–185, Jun. 2016.

[24] A. Onan, S. Korukoglu, and H. Bulut, "LDA-based topic modelling in text sentiment classification: An empirical analysis," *Int. J. Comput. Linguistics Appl.*, vol. 7, no. 1, pp. 101–119, 2016.

[25] H. Qian, Y. Mao, W. Xiang, and Z. Wang, "Recognition of human activities using SVM multi-class classifier," *Pattern Recognit. Lett.*, vol. 31, no. 2, pp. 100–111, 2010.

[26] C. Shen, T. Li, and C. H. Ding, "Integrating clustering and multi-document summarization by bi-mixture probabilistic latent semantic analysis (PLSA) with sentence bases," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, pp. 914–920.

[27] P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2009, pp. 442–457.

[28] E. M. Tapia, S. S. Intille, and K. Larson, "Activity recognition in the home using simple and ubiquitous sensors," in *Proc. Int. Conf. Pervasive Comput.*, 2004, pp. 158–175.

[29] S. Tata and J. M. Patel, "Estimating the selectivity of tf-idf based cosine similarity predicates," *ACM SIGMOD Rec.*, vol. 36, no. 2, pp. 7–12, 2007.

[30] Y. Tian, Q. Yang, T. Huang, C. X. Ling, and W. Gao, "Learning contextual dependency network models for link-based classification," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1482–1496, Nov. 2006.

[31] T. Tietz, J. Waitelonis, J. Jäger, and H. Sack, "refer: A linked data based text annotation and recommender system for wordpress," *presented at the ISWC Posters Demonstrations Track*, Kobe, Japan, Oct. 19, 2016.

[32] T. Van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, "Accurate activity recognition in a home setting," in *Proc. 10th Int. Conf. Ubiquitous Comput.*, 2008, pp. 1–9.

[33] D. Wang, H. Zhang, and R. Liu, et al., "Feature selection based on term frequency and t-test for text categorization," in *Proc. 21st ACM Int. Conf. Inf. Knowledge Manag.*, CIKM'12, Maui, HI, USA, Oct./Nov. 2012, pp. 1482–1486.

[34] Y. Wang, et al., "Regularity and conformity: Location prediction using heterogeneous mobility data," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1275–1284.

[35] Z. Wu, Y. Xu, C. Zhang, Y. Yang, and Y. Ji, "Towards Semantic Web of things: From manual to semi-automatic semantic annotation on web of things," in *Proc. Int. Conf. Big Data Comput. Commun.*, 2016, pp. 295–308.

[36] Z. Xu, R. Jin, K. Huang, M. R. Lyu, and I. King, "Semi-supervised text categorization by active search," in *Proc. 17th ACM Conf. Inf. Knowl. Manage.*, 2008, pp. 1517–1518.

[37] X.-B. Xue and Z.-H. Zhou, "Distributional features for text categorization," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 3, pp. 428–442, Mar. 2009.

[38] L. Yang, X. Cao, D. Jin, X. Wang, and D. Meng, "A unified semi-supervised community detection framework using latent space graph regularization," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2585–2598, Nov. 2015.

[39] L. Yao and Q. Z. Sheng, "Correlation discovery in web of things," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 215–216.

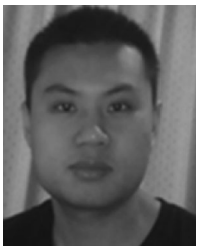
[40] L. Yao, Q. Z. Sheng, B. J. Gao, A. H. Ngu, and X. Li, "A model for discovering correlations of ubiquitous things," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2013, pp. 1253–1258.

[41] H. Yu, J. Yang, and J. Han, "Classifying large data sets using SVMs with hierarchical clusters," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 306–315.

[42] S. Zelikovitz and F. Marquez, "Transductive learning for short-text classification problems using latent semantic indexing," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 19, no. 02, pp. 143–163, 2005.

[43] R. Zhang, L. Zhang, X.-J. Wang, and L. Guan, "Multi-feature pLSA for combining visual features in image annotation," in *Proc. 19th ACM Int. Conf. Multimedia*, 2011, pp. 1513–1516.

[44] Z. Zheng, Y. Chen, T. He, F. Li, and D. Chen, "Weight-RSS: A calibration-free and robust method for WLAN-based indoor positioning," *Int. J. Distrib. Sensor Netw.*, vol. 2015, 2015, Art. no. 55.



Yuanyi Chen received the BSc degree from Sichuan University, in 2010 and the master's degree from Zhejiang University, in 2013. From September 2014 to September 2015, he was a jointly-supervised PhD candidate at Hong Kong Polytechnic University. He is working toward the PhD degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research topic includes the Internet of Things and mobile computing.

Jingyu Zhang received the BSc degree from Hunan Normal University, in 2008 and the ME degree from Chongqing Jiaotong University, in 2011. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include mobile computing, computer architecture, cache optimization, and adaptive video streaming.

Liting XU received the BSc and master's degrees from Jiangsu University, China, in 2005 and 2007, respectively, both in computer science and applications. During 2008–2011, she was a Software R&D engineer with TERAOKA Weigh-System Pte. Ltd, Singapore. From 2011 to 2012, she was with ASM Technology Singapore Pte Ltd, as a software R&D engineer. After that, she had been a software modelling engineer in Continental Automotive Singapore Pte Ltd. In May. 2013, she joined Shanghai Jiao Tong University, China, as research engineer. Her present research interests include computer supported cooperative work (CSCW), applications of distributed computing, and web visualization on the Grid.

Minyi Guo received the BS and ME degrees in computer science from Nanjing University, China, in 1982 and 1986, respectively, and the PhD degree in information science from University of Tsukuba, Japan, in 1998. From 1986 to 1994, he had been an assistant professor of the Department of Computer Science, Nanjing University. He is currently a chair professor and the head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include parallel computing, distributed computing, and mobile computing. He has more than 300 publications in major journals and international conferences in these areas. He is on the editorial board of the journals the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He is a senior member of the IEEE, a member of the ACM, IEICE IPSJ, and CCF.

Jiannong Cao received the BSc degree from Nanjing University, China, in 1982, and the MSc and PhD degrees from Washington State University, Washington, in 1986 and 1990, respectively, all in computer science. He is currently a chair professor and the head of the Department of Computing, Hong Kong Polytechnic University. His research interests include parallel and distributed computing, computer networks, mobile and pervasive computing, fault tolerance, and middleware. He co-authored four books, coedited nine books, and published more than 300 technical papers in major international journals and conference proceedings. He has directed and participated in numerous research and development projects and, as a principal investigator, obtained over HK$25 million grants. He is the chair in Technical Committee on Distributed Computing, IEEE Computer Society. He has served as an associate editor and a member of editorial boards of many international journals, and a chair and a member of organizing/program committees for many international conferences. He is a fellow of the IEEE, a member of the ACM, and a senior member of the China Computer Federation.

Providing Service Continuity in Clouds Under Power Outage

Weiwei Wu, *Member, IEEE*, Jianping Wang, *Member, IEEE*, Kejie Lu, *Member, IEEE*, Wen Qi, Feng Shan, *Member, IEEE*, and Junzhou Luo, *Member, IEEE*

Abstract—In cloud computing, it is crucial to maintain service continuity, while power outage is one of the most common and serious threats. To improve the resilience of cloud against power outage, a service provider usually deploys emergency energy supply (e.g., UPSs and generators) in a data center. When a power outage at a data center happens, the cloud service provider needs to make the operation decision on which subset of VMs to keep running and which servers to host such VMs to minimize its loss (or maximize its profit) using the emergency energy supply while the selected VMs are running in the affected data center until they are finished, migrated to other data centers, or normal power supply of the affected data center has been restored. No prior research has theoretically studied such a cloud service continuity problem under power outage. In this paper, we tackle this challenge and investigate the cloud service continuity problem. Specifically, we consider that a profit is associated with maintaining the continuity of a service, denoted as *service continuity profit*. Based on that we first formulate an optimization problem that aims to maximize the total profit subject to energy constrains. After showing the hardness of the problem, we focus on the design of approximation algorithms for solving the problem, where we consider two practical cases. In the first one with sufficient number of servers for re-provisioning, we develop a constant approximation algorithm of which the worst-case performance approaches the optimal solution within a constant factor ($\approx$ 4.5-6.4). In the second one, we consider the general case with limited number of servers, and we develop an approximation algorithm with an approximation ratio of around 5.7-8. By combining these two algorithms together, we can achieve both good worst-case performance and average performance. Simulation results demonstrate the efficiency in terms of maximizing the service continuity profit of the proposed algorithms.

Index Terms—Service continuity, power outage, cloud recovery, VM consolidation, energy-efficient scheduling, approximation algorithm, profit maximization

1 Introduction

Despite the salient features of cloud computing, cloud services may be interrupted due to various issues. As reported in [1], [2], data centers across the world suffer frequently from the outages and costs $600,000 per incident on average. Among the root causes of those outages, the top one is power outage [1]. For example, Amazon Web Services suffered an approximately 4-hour power outage in its North Virginia data center from 8:50 p.m. to 1:09 a.m. on June 14, 2012, causing major disruption to numerous web companies that rely on Amazon's cloud service, including Instagram, Netflix and Pinterest [3]. In 2015, a cloud data center in Hong Kong that is part of Alibaba Ltd., the biggest e-commence company of China, also suffered a 14-hour disruption due to power outage from 9:37 a.m. to 11:39 p.m. on June 21 [4].

Clearly, service interruption has significant negative impacts on both the cloud provider and customers, especially for the critical services sensitive to interruption. To improve the resilience of cloud services against power outage, there are two main approaches. First, a common practice in the cloud industry is to install emergency energy supply, such as UPSs and generators. Second, since a cloud service provider usually owns distributed data centers, it can migrate some services to other data centers that are not affected by power outage, while migration time varies among different services as it takes time to launch a Virtual Machine (VM) image, migrate storage and establish network connection. Based on these strategies, to maintain the continuity of a service, the VM supporting the service needs to keep operating locally until the minimum time point, named deadline, among the time points at which the service is finished, the VM supporting the service is successfully migrated to the new data center, or the normal power supply of the affected data center is back, which can be considered as the *continuity requirement*.

Although the approaches above are viable, we note that there is a lack of a theoretical study in the literature that addresses how to optimally exploit emergency energy and external data centers during power outage. In this paper, we tackle this challenging issue and investigate a *cloud*

- *W. Wu, F. Shan, and J. Luo are with the School of Computer Science and Engineering, Southeast University, Nanjing, Jiangsu 210096, P.R. China. E-mail: {weiweiwu, shanfeng, jluo}@seu.edu.cn.*
- *J. Wang and W. Qi are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: jianwang@cityu.edu.hk, qi.wen@my.cityu.edu.hk.*
- *K. Lu is with the Department of Electrical and Computer Engineering, University of Puerto Rico at Mayaguez, Mayagüez, PR 00682, and the School of Computer Engineering, Shanghai University of Electric Power, Shanghai 200090, China. E-mail: kejie.lu@upr.edu.*

Manuscript received 29 Jan. 2017; revised 6 June 2017; accepted 3 July 2017. Date of publication 19 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Jianping Wang.) Digital Object Identifier no. 10.1109/TSC.2017.2728795

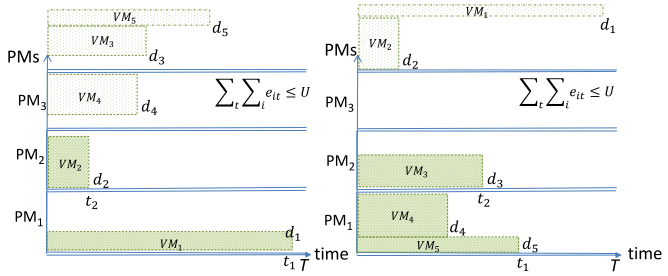


Fig. 1. Two exemplary schedules for providing service continuity. Each rectangle represents a VM with service continuity requirement, of which the height is its demand of resource and the width is the time period of continuity requirement required for running VM locally before the service is finished or transferred. The light-colored ones fail to keep their service continuity due to the lack of emergency energy.

service continuity (CSC) problem under power shortage. Specifically, we consider that each service has a specific continuity requirement, and we define a *service continuity value/profit* associated with each VM/service if its continuity requirement is satisfied. Since the amount of emergency energy supply is limited, the CSC problem is an optimization problem whose objective is to maximize the total service continuity profit by identifying (VM selection) and re-provisioning a subset of existing VMs to *physical machines* (PMs) in the local data center, subject to the resource requirements of VMs, the resource capacity of PMs, as well as the limitation of emergency energy supply.

Now we use an example shown in Fig. 1 to illustrate that inefficient utilization of emergency energy may result in high loss of service continuity profit. In this example, when power outage strikes, there are five VMs requiring their service continuity to be satisfied with the support of an amount U of emergency energy under power outage. Each VM is represented as a rectangle where the height is its demand of resource (e.g., CPUs) and the width is the time period of continuity requirement, during which it needs to keep running until the service hosted by the VM is finished or transferred. Suppose simply that there are three PMs in total, each with an identical capacity of resource. With limited amount of emergency energy, the first schedule chooses/attempts to maintain the service continuity of VM_1 and VM_2 by consolidating them to the first two PMs, each with one VM consolidated, where e_{it} is the power consumption of PM_i at time t. Suppose that the profit for keeping each VM's service continuity is approximately the size of its area. Then, another schedule that chooses VMs with similar width (VM_3, VM_4, VM_5) and attempts to fully utilize the resource/capacity of PMs, say, the second schedule, would gain more overall profit of service continuity under the same budget constraint of emergency energy. Compared with the second schedule, the first one spends too much power on maintaining the continuity of VM_1 and keeping PM_1 active, but fails to efficiently utilize the resource as well as the power over time. Therefore, there is a need for designing efficient scheduling strategy to maintain service continuity with maximum profit under power outage.

In the literature, such a CSC problem has not been investigated before but there are some related studies. One of the most relevant work is VM consolidation [5], [6], with which VMs can be re-provisioned and idle PMs can be shut down to reduce the power consumption. Although VM consolidation has been studied extensively in the past, most of the studies focus on minimizing the operational cost of a single data center (e.g., [7], [8], [9], [10]) or geo-distributed data centers (e.g., [11], [12], [13]). Moreover, all these prior work have assumed unlimited power supply which can support all VMs indefinitely, thus have no need to study VM selection strategy. This is different from CSC problem that exploits the limited emergency energy and needs to design both VM selection strategy and VM consolidation strategy under power shortage so as to maximize the total service continuity profit. Besides VM consolidation, there are some other related work concerning about cloud recovery, but few works have theoretically addressed the scheduling problem on providing service continuity during power outage. A full review can be referred to in Section 2. To summarize, we note that existing studies cannot be applied to solve our CSC problem which has both different objectives and constraints.

To fully utilize the limited emergency energy, ideally if there are sufficient number of PMs for re-provisioning, we shall select and re-provision VMs to servers according to their service continuity requirements, i.e., time period during which they require to keep running in the affected data center. By doing this, VMs on a server could finish their services around the same time and then that server can be shut down. If the number of PMs for re-provisioning is limited, VMs with various service continuity requirements may have to be packed together, which will prolong the service time of such servers under power shortage. In this paper, we consider both aforementioned scenarios and develop algorithms with constant approximation ratios to maximize the service continuity profit.

The contributions of this paper are summarized as follows.

- This paper studies the service continuity problem for cloud data centers under power outage. This is the first work to theoretically study scheduling algorithms to exploit the emergency energy and maximize the profit of keeping service continuity in cloud data centers under power shortage.
- When the number of PMs for re-provisioning is sufficiently large, we propose a constant approximation algorithm that achieves a profit within around 4.5-6.4 times of the optimal solution. We first propose a procedure to get a bundle of VMs with high total profit, and then apply it iteratively to re-provision each bundle of VMs to a PM so as to gain high aggregated profit of service continuity.
- When the number of PMs for re-provisioning is limited, we propose a constant approximation algorithm (with an approximation ratio $\approx$ 5.7-8). We introduce a novel fractional version of the problem with well-organized optimal structures and transform its optimal solution to be a feasible solution of CSC problem with small loss of approximation.
- Finally, the two algorithms above are extended and combined to be a service continuity strategy that can achieve a good average performance, as well as a theoretical worst-case bounded performance. Simulation results verify that the average performance of the proposed strategy is close to the optimal profit that is achievable for the optimal solution.

The organization of the paper is as follows. Section 2 reviews the related work. Section 3 formulates the problem and provides the overview of our solutions. Section 4 develops an approximation algorithm for the situation that the number of PMs for re-provisioning is sufficiently large. Section 5 studies the general case with the limitation on the number of PMs. Numerical results are presented in Section 6. We conclude the paper in Section 7.

2 Related Work

Much research effort has been devoted in developing energy-efficient scheduling algorithms in data centers [14]. We just review the ones most related to the problem/technique considered in this paper.

One of the techniques most related to the one adopted in this paper is VM consolidation [5], [6], with which VMs can be re-provisioned and idle PMs can be shut down to the efficiency of energy usage. Although VM consolidation has been studied extensively in the past, most of the studies focus on minimizing the operational cost of a single data center (e.g., [7], [8], [9], [10]) or geo-distributed data centers (e.g., [11], [12], [13]). Rich research works focus on optimizing the data center operation such as energy efficiency and quality of service (QoS). Meisner et al. [15], Lin et al. [16] reduce the power consumption by reducing the number of idle servers. The works in [17], [18] leverage energy storage devices to avoid high electricity usage when the electricity price increases. Xu [19] proposes an idea of using partial execution to reduce the peak power demand and energy cost of data centers. Lin et al. [7], Liu et al. [20], Lim et al. [21] consider parameterized optimization objectives to minimize the energy cost and response time. Xu et al. [22] proposes to make temperature-aware workload management for geo-distributed data centers. Xu and Liang [23], Qureshi et al. [24] consider geo-graphical electricity price diversity to reduce the operation cost of data centers. More related works can be referred to in survey papers [5], [6].

The service continuity problem studied in this paper needs to exploit the emergency energy and provide service continuity for a subset of VMs, that is carefully selected, to maximize the total profit of service continuity with the support of limited emergency energy. We note that a VM consolidation technique can be resorted to enhance the efficiency of emergency energy usage. However, all existing works in the literature of VM consolidation introduced above have assumed unlimited power supply, which can support all VMs indefinitely, and aim at minimizing the operation cost with the support of infinite energy supply. They cannot be applied to our scenario that exploits the limited emergency energy and needs to design both VM selection strategy and VM consolidation strategy under power short age, so as to maximize the total profit of keeping service continuity.

Prior to our work, only a few works have discussed the problem on cloud recovery or providing service continuity. Wood et al. [25] discuss about the possibility of providing the disaster recovery as a service in the cloud. Klems et al. [26] provides a solution to the recovery of IT services in the event of a disaster with the help of the cloud. Develder et al. [27] and Habib et al. [28] exploit the backup scheme to provide the resilience or protection ability in case of link or server failure. Few prior works have theoretically addressed the restoration scheme or the problem on how to schedule the VMs and provide service continuity with limited emergency energy under power outage.

In summary, we note that existing studies cannot be applied to solve the service continuity problem under consideration. To the best knowledge of the authors, this paper is the first work to theoretically study the scheduling problem on providing service continuity with maximum profit in cloud data center under power shortage.

3 Problem Formulation and Overview of Our Solution

In this section, we formulate the cloud service continuity problem and provide the overview of our solutions.

3.1 System Model

Suppose the power supply outside the cloud data center is interrupted at time 0 due to a disaster, starting from which the only available power is the emergency energy (such as UPSs, power generator) with limited energy, and the outside power supply is expected to be restored at time T.

When the outside power supply is cut off, the cloud service provider can redirect some services to other data centers that are not affected by power outage. Since cloud services are realized by Virtual Machines (VMs), to maintain the continuity of a service, the VM supporting the service needs to keep operating locally until the time point, named deadline, at which the service is finished or the VM supporting the service is successfully launched in the new data center and ready for redirection. Let $J = \{1, 2, \ldots, n\}$ be the set of VMs that is running in the cloud data center before power outage occurs. To facilitate the scenario and capture the fundamental challenge in maintaining service continuity under power shortage, we measure the resource demand of the VMs by the request of CPUs or vCPUs (virtual CPUs) since the CPU usage usually takes up a significant share of the total power needed. Each VM j is assumed to have a *size/demand* s_j, requesting s_j units of CPUs or vCPUs. When power outage strikes, the VM supporting the service needs to keep operating locally until the minimum time point, named deadline, among the time points at which the service is finished, the VM supporting the service is successfully migrated to the new data center not affected by the power outage or the normal power supply is back. Let d_j be the deadline of VM j, which specifies a period $[0, d_j)$ of *continuity requirement* that is used to keep the VM operating and maintaining the service continuity. Note that such a continuity requirement can include both the time required for running VMs and the time caused by migrating the VMs (either to another PM or to the remote data center).

Thus, to satisfy its continuity requirement, VM j needs s_j units of resource (resource demand) in time interval $[0, d_j)$ to keep operating. Assume that it contributes a value/profit p_j if its service continuity requirement is satisfied. Here, profit p_j will be called *service continuity profit* of VM j and deadline d_j will be called the *length* of VM j. We assume naturally that no new VM demands will be accepted during the period $[0, T]$ of the power outage, thus all VMs are available at time 0.

Let $I = \{1, 2, \ldots, m\}$ be the set of *physical machines* in the cloud data center. Each PM is assumed to be identical, each with the same capacity V of resource.

3.2 Problem Formulation

Due to the lack of emergency energy during power outage, not all VMs can keep operating until their deadlines of continuity requirements, thus the service continuity requirements of some VMs may be violated. It is necessary to design both a VM selection strategy and a VM consolidation strategy so as to efficiently utilize the emergency energy and maximize the total service continuity profit of the VMs for the cloud provider.

A re-provision/consolidation strategy would consolidate the VMs to the PMs and shut down idle machines to efficiently utilize the energy. Define x_{ij} as an indicator to indicate if VM j is allocated/re-provisioned to PM i. That is, $x_{ij} = 1$ if VM j is executed on PM i and $x_{ij} = 0$ otherwise. Thus, we have

$$x_{ij} = \{0, 1\}, \quad \forall 1 \leq i \leq m, \forall 1 \leq j \leq n. \tag{1}$$

Due to the lack of emergency energy and the constraints of resource capacities, some VMs have to be interrupted before their deadlines, thus fail to keep the service continuity. That is, it is possible that $\sum_{1 \leq i \leq m} x_{ij} = 0$ for some VM j. In this case we say that they fail to be allocated/consolidated. Thus, before designing a consolidation strategy under power shortage, it is critical to determine a VM selection strategy: determine which set of VMs should be chosen and consolidated (with $\sum_{1 \leq i \leq m} x_{ij} = 1$).

After determining the VM selection, an efficient consolidation strategy under power shortage is required to consolidate the selected VMs by efficiently utilizing the emergency energy. Here, each VM j is executed/consolidated on at most one PM

$$\sum_{1 \leq i \leq m} x_{ij} \leq 1, \quad \forall 1 \leq j \leq n. \tag{2}$$

Eqs. (1) and (2) are called the *assignment constraints*. The number of PMs activated for re-provisioning/consolidation should be at most m, which is called the *PM constraint*.

Because of the resource constraints of PMs, the total demands of VMs consolidated on PM i must not exceed V in total. That is,

$$\sum_{1 \leq j \leq n} s_j x_{ij} \leq V, \quad \forall 1 \leq i \leq m. \tag{3}$$

This is called the *resource capacity constraint*. We say that VM j *fits* PM i if it can be allocated to PM i without exceeding the resource capacity.

Before modeling the power constraint, we first introduce the power consumption model adopted in this paper. Let $u_{it} \in [0, 1]$ be the CPU utilization rate of machine i at time t,

$$u_{it} = \sum_{j: t \in [0, d_j)} s_j x_{ij} / V, \quad \forall t, \forall i. \tag{4}$$

We say PM i is *active* at time t if $u_{it} > 0$, thus it uses one active/power-on PM time unit. As measured in prior works [15], [16], the power consumption of a PM approximately follows the function below,

$$e_{it} = \begin{cases} (1 - \alpha) u_{it}^{\mu} + \alpha, & \text{if } u_{it} > 0 \\ 0, & \text{if } u_{it} = 0, \end{cases} \tag{5}$$

where $\mu \geq 1$ and the peak power with full CPU utilization is normalized to be 1; the constant α is the idle power, which is typically ranging in $[0.5, 0.7]$ (and barely below than 0.5) as noted in practical measurements [15], [29], [30].

We measure the migration cost as the migration time used for migrating a VM to a new PM, since this term affects the energy overhead of migration as well as the migration latency, as measured in [31]. Note that instead of explicitly formulating the migration time of a VM, we have incorporated it into the time period $[0, d_j)$ of continuity requirement of each VM, as introduced in the definition of continuity requirement.

We assume that there are U units of emergency energy when power outage strikes. Equivalently, it is able to support a PM with full CPU utilization to execute U time units after normalization. A schedule should satisfy the *power constraint* that the total power consumed should be at most U,

$$\sum_{1 \leq i \leq m, 1 \leq t \leq T} e_{it} \leq U. \tag{6}$$

Obviously, PM i must be powered on until all assigned VMs finish their execution. Let t_i be the length/number of power-on time of PM i, then it should be no less than the required running time of any VM consolidated on the current PM. We thus have

$$t_i \geq d_j x_{ij}, \quad \forall 1 \leq j \leq n, \forall 1 \leq i \leq m. \tag{7}$$

For VM j, if its service continuity requirement is satisfied (with $\sum_{1 \leq i \leq m} x_{ij} = 1$), then the cloud service provider gains a *value/profit* p_j from VM j; otherwise, it gains a profit 0. If a VM is consolidated in the service continuity schedule, then we say the schedule *executes/starts* the VM, interchangeably.

Our objective is to maximize the aggregated profit of VMs whose service continuity requirement is satisfied, i.e.,

$$\sum_{1 \leq j \leq n} \left(p_j \sum_{1 \leq i \leq m} x_{ij} \right). \tag{8}$$

Therefore, the cloud service continuity problem under consideration is summarized in the following definition (ILP formulation).

Definition 1 (CSC problem). *The Cloud Service Continuity problem is to determine the allocation $\{x_{ij} | i \in I, j \in J\}$ to maximize the service continuity profit (8) with the satisfaction of the assignment constraint (1) and (2), the resource capacity constraint (3), the power constraint (6) and the constraint (7) and the PM constraint.*

We investigate the CSC problem from the approximation point of view. We say that an algorithm is γ-approximation if it always achieves a profit within $\frac{1}{\gamma}$ times that of the optimal solution for any input/instance I. That is,

$$\frac{ALG(I)}{OPT(I)} \geq \frac{1}{\gamma}, \ \forall I. \tag{9}$$

where $ALG(I)$ and $OPT(I)$ are respectively the service continuity profit achieved by the algorithm and the optimal solution.

3.3 Overview of Our Solutions

In CSC problem, we need to select and re-provision/consolidate the VMs to exploit the emergency energy and maximize the service continuity profit of VMs whose continuity requirements are satisfied using limited emergency energy.

The CSC problem (even consolidating with unlimited number of PMs) can be easily proved to be NP-hard by reducing the classical NP-hard knapsack problem to an instance of it. Therefore, in this paper, we focus on the design of approximation algorithms.

The high level idea of the design is as follows. We observe that the idle power α is a large constant compared with the peak power and a PM using U active PM time units spends at most U units of peak power (since it is normalized to be 1). Thus, algorithms that uses at most U active PM time units guarantee to satisfy the power constraints.

We first consider the situation that the number of PMs available for re-provisioning is sufficiently large in Section 4. This allows us to activate a new PM if necessary. We develop an algorithm Bundle Re-provision (BRP) to maximize the service continuity profit of the provider. BRP ensures that each bundle of VMs assigned to one PM is with high profit per unit of power-on time and is proved to be $\frac{2e}{(1-\delta)(e-1)\alpha}$-approximation compared to the optimal solution.

Next, in Section 5, we consider the general case that the number of PMs available for re-provisioning has a limit and develop a final algorithm, called CSC-SCHEDULE, by combining two efficient algorithms introduced below to achieve a good average performance as well as a worst-case bounded performance.

With the restriction on the number of PMs for re-provision, the PMs should be carefully activated to ensure that each activated PM gains high total profit by allocating the VMs. One intuitive idea to deal with the new constraint is to extend BRP by greedily activating the PMs until all m PMs are used up. This revised algorithm is called BRP*. In general, BRP* has a good average performance. However, this may cause the problem that the activated PM gains low total profit when all chosen VMs have short lengths, although the profit per power-on time unit is high. As a matter of fact, this may make the worst-case performance be arbitrarily bad.

Thus, we combine BRP* with another algorithm, called FRP* (Fractional Re-provision), which will be proved constant approximation in terms of worst-case performance. In the design of FRP*, we first introduce a fraction version of CSC problem with nice optimal structures and develop a novel algorithm to find its optimal solution, and then we transform this solution back to be a feasible solution of the original problem with a loss of small approximation ratio. By returning the better one between BRP* and FRP*, we have the final schedule CSC-SCHEDULE.

4 Algorithm Design with Sufficient Number of PMs

In this section, we consider CSC problem in clouds under a natural condition that the number of PMs available for re-provisioning is sufficiently large. Since a PM using U active PM time units guarantees to spent at most U peak power, we introduce a *time-constrained CSC problem* first where the original power constraint is replaced by the *time-constraint* that the total number of power-on time units is not allowed to exceed U. We will devise an algorithm for the time-constrained CSC problem, which tries to fully utilize the CPU resource at each unit of power-on time and meanwhile guarantees to satisfy the power constraints in the original CSC problem. Then, we will prove that the proposed algorithm is close to the optimal profit in time-constrained CSC problem, which will be further proved to approach the optimal solution with the original power constraints.

Here, a sufficient number of PMs does not mean that it is free to activate as many as wanted, due to the lack of emergency power. Thus, the VMs need to be carefully selected and consolidated to PMs to gain the maximum profit.

One intuitive and promising idea is to design a greedy algorithm that allocates the VMs in the order of decreasing profit-per-deadline and adopts first-fit strategy to allocate the VM to the first active PM and activate one new PM if no active PMs are available. However, a simple example can be easily found to show that such a greedy algorithm has unbounded approximation ratio, since it does not take the resource capacities of PMs and profit diversity of VMs into consideration, which may lead to the waste of resource of a PM as well as the waste of power or power-on time in the long run.

Our basic idea for solving the CSC problem is to consider a key sub-problem to guarantee the efficient usage of each power-on time unit in activating PMs and then adopt it as a building block to activate new PMs. The key sub-problem under consideration is, given a budget of U units of power-on time, maximizing the profit of VMs chosen to be executed in one single PM.

The arrangement of this section is as follows. We first develop a basic procedure to solve the key sub-problem above. Then, we propose an algorithm for the CSC problem by iteratively calling the basic procedure. Finally, we analyze the the proposed algorithm and prove its constant approximation.

4.1 A Basic Procedure

In this section, given t power-on time units, we examine the key sub-problem on how to select VMs from J to run on a single PM with capacity V during $[0, t)$, so that the total profit is maximized. Define $J^{\leq t} \subseteq J$ to be the set of VMs whose deadlines satisfy $d_j \leq t$. Obviously, only VMs with deadlines not greater than t can start on such a PM. Thus, we only need to consider the set $J^{\leq t}$.

We use a basic procedure to solve the problem above. This procedure will find a subset of VMs from a given set $\hat{J}$ such that the total profit is maximized with the constraint that the total resource demands of the selected VMs is at most V. Given a set $\hat{J}$ of VMs and a capacity V, let procedure PROFITABLEBUNDLE($V, \hat{J}$) return the desired subset and $profit(V, \hat{J})$ be the maximum profit.

Note that such a sub-problem can be reduced to be the transitional knapsack problem, thus it admits a dynamic programming algorithm. Let $p(j, v)$ be the maximum profit achievable using only the first j VMs in $\hat{J}$ with a capacity v. The profit $p(j, v)$ is either achieved by $p(j-1, v)$ when the jth VM is not selected or by $p(j-1, v-s_j)+p_j$ when the jth VM is selected. Thus, the recursion function is

$$p(j,v) = \max\{p(j-1,v), \\ p(j-1,v-s_j)+p_j\}. \tag{10}$$

Based on the recursion function above, we can implement ProfitableBundle($V, \hat{J}$) by computing $p(|\hat{J}|, V)$. The time complexity is $O(nV)$ and pseudo-polynomial due to its dependency on value V. If needed, we can further remove the dependency on value V of the time complexity by introducing a pre-set small error factor δ, a value/constant that can be arbitrarily close to 0, so as to obtain a total profit arbitrarily close to $p(|\hat{J}|, V)$ within a factor of $(1-\delta)$ (namely, a fully polynomial-time approximation scheme, FPTAS), using $O(n^3 \frac{1}{\delta})$ running time given any error bound δ. The detailed implementation is similar to the FPTAS for the well-known knapsack problem [32], thus is omitted here.

To solve the key sub-problem, we only need to invoke procedure ProfitableBundle($V, J^{\leq t}$), where $J^{\leq t}$ is the set of VMs whose deadlines satisfy $d_j \leq t$.

4.2 Algorithm Design

Based on the solution above, we are now ready to solve the CSC problem with sufficient number of PMs for re-provisioning.

The idea of the design is to activate one single PM to run a bundle of VMs (with the largest possible total profit per unit of power-on time) first at each iteration. And then, we iteratively activate a new PM while satisfying the power-on time-constraints as well as the power constraints. The intuition behind such a general idea is to efficiently utilize the resource capacity and achieve high profit when consuming each power-on time unit.

In detail, the first bundle of VMs is computed as follows. With the help of algorithm ProfitableBundle, we compute the maximum profit $profit(V, \hat{J}^{\leq t})$ given any duration $[0, t)$. The first bundle is the set of VMs that achieves the maximum value $\frac{profit(V,\hat{J}^{\leq t})}{t}$ among all possible integer $t \in [0, \min\{T, U\}]$. Then, the algorithm iteratively finds the rest of the bundles.

The detailed design of the algorithm is presented in BRP. Let A_k be the subset of VMs allocated in iteration k and $S_k = \cup_{1 \leq i \leq k} A_i$. In iteration k, the algorithm calls the procedure ProfitableBundle($V, (J \backslash S_{k-1})^{\leq t}$) to test every time $1 \leq t \leq \min\{T, U\}$ to find a value $\hat{t} = \arg\max_t \frac{profit(V,(J\backslash S_{k-1})^{\leq t})}{t}$ and set A_k to be the VMs that achieve $profit(V, (J \backslash S_{k-1})^{\leq \hat{t}})$. It activates the kth PM to allocate VMs in A_k and gets profit $profit(V, (J \backslash S_{k-1})^{\leq \hat{t}})$ with the cost of an amount E_k of power consumption. The algorithm iteratively finds the next set S_{k+1} and terminates either when all VMs are chosen or the total energy consumption exceeds the capacity U after adding the set S_{k+1}. Should this case occur, the algorithm returns the set with higher profit between S_k and $\bar{S}$, where $\bar{S}$ is the maximum profit achievable for one single PM with the input of VMs in J and U time units.

In terms of the time complexity, computing S_k would call the procedure ProfitableBundle() at most $\min\{T, U\} \cdot \min\{m, U\}$ times, while computing $\bar{S}$ would call the procedure ProfitableBundle() at most $\min\{T, U\}$ times, where typically we have $U > T$ and $U > m$ when power outage happens. Thus, the running time of BRP is $O(mTnV)$ when setting $\delta = 0$.

4.3 Approximation Ratio Analysis

Now we analyze the performance of the Algorithm BRP. Recall the definition of the time-constrained CSC problem and denote by OPT^c the subset of VMs scheduled by the optimal solution of the time-constrained CSC problem using at most U units of power-on time, and let $p(OPT^c)$ be the total profit of those VMs. Denote by $p(OPT)$ the optimal profit of the optimal solution in the original CSC problem. We first prove that the algorithm achieves within $\frac{2e}{(1-\delta)(e-1)}$ times of the optimal profit in time-constrained CSC problem where e is the natural constant and δ is the constant error factor introduced to bound the performance on the total profit.

Algorithm 1. BRP(V, J, m)

```
Set A_0 = S_0 = ∅
Set k = 1, t̂ = 0
while J\S_{k-1} ≠ ∅ and U > 0 do
    Compute t̂ = arg max_{1≤t≤min{T,U}} profit(V,(J\S_{k-1})^{≤t}) / t
    if U − t̂ > 0 then
        A_k = ProfitableBundle(V, (J\S_{k-1})^{≤t̂})
        Activate the kth PM to run the VMs in A_k.
        S_k = S_{k-1} ∪ A_k.
        Compute the amount of power, say E_k, consumed on
        on the kth PM.
        k = k + 1, U = U − E_k.
    end if
end while
Compute t̄ = arg max_{1≤t≤{T,U}} profit(V, J^{≤t})
Let S̄ = ProfitableBundle(V, J^{≤t̄}).
if the profit in S_k is larger than that of S̄ then
    return the profit in S_k.
else
    return the profit in S̄.
end if
```

Suppose that the algorithm finishes in p iterations and the algorithm returns set S_p at termination. Let A_k be the VMs returned in the kth iteration, and $p(A_k)$ be the total profit of VMs in A_k, where $1 \leq k \leq p$.

We first prepare a basic property for the greedy rule of the algorithm. The following lemma states the fact that each bundle of VMs selected in the iterations is profitable.

Lemma 1. *$p(S_k) - p(S_{k-1}) \geq (1-\delta) t_k \cdot \frac{p(OPT^c \backslash S_{k-1})}{U}$ for all $1 \leq k \leq p$.*

Proof. Let t_k be the largest deadline of VMs in A_k. Let $opt(t, J)$ be the maximum achievable profit to run the VMs in J by activating only one PM with t time units. For the first iteration, assuming t' is the value that achieves $opt(t', J)\frac{1}{t'} = \max_t opt(t, J)\frac{1}{t}$, we have $p(A_1)\frac{1}{t_1} = \max_t profit(t, V, J)\frac{1}{t} = profit(t', V, J)\frac{1}{t'} \geq (1-\delta) opt(t', J)\frac{1}{t'}$ where the last inequality holds by the fact that ProfitableBundle returns a subset with at least a profit $profit(t', V, J) \geq (1-\delta) opt(t', J)$. Moreover, $opt(t', J)\frac{1}{t'} = \max_t opt(t, J)\frac{1}{t} \geq p(OPT^c)\frac{1}{U}$ since $opt(t', J)\frac{1}{t'}$ is the maximum profit per power-on time that is achievable, and activating any PM in the optimal solution OPT^c of time-constrained CSC problem achieves at most the profit $opt(t', J)\frac{1}{t'}$ per unit time. Therefore, combining these inequalities, we have $p(A_1)\frac{1}{t_1} \geq (1-\delta) opt(t', J)\frac{1}{t'} \geq (1-\delta) p(OPT^c)\frac{1}{U}$.

Let t' be the value that achieves $opt(t', J \backslash S_{k-1})\frac{1}{t'} = \max_t opt(t, J \backslash S_{k-1})\frac{1}{t}$. Then, in later iteration k, we have

$(p(S_k)-p(S_{k-1}))\frac{1}{t_k}=\max_t profit(t,V,J\backslash S_{k-1})\frac{1}{t}=profit(t',V,J\backslash S_{k-1})\frac{1}{t'}\geq(1-\delta)opt(t',J\backslash S_{k-1})\frac{1}{t'}$ where the first inequality holds because of the greedy rule in the kth iteration applied in the algorithm and the last inequality is correct because the procedure ProfitableBundle returns a subset with at least a profit $profit(t',V,J\backslash S_{k-1})\geq(1-\delta)opt(t',J\backslash S_{k-1})$. Moreover, $opt(t',J\backslash S_{k-1})\frac{1}{t'}=\max_t opt(t,J\backslash S_{k-1})\frac{1}{t}\geq p(OPT^c\backslash S_{k-1})\frac{1}{U}$, since $opt(t',J\backslash S_{k-1})\frac{1}{t'}$ is the maximum profit per cost that is achievable among the VMs $J\backslash S_{k-1}$ and activating any PM in OPT^c achieves at most the profit $opt(t',J\backslash S_{k-1})\frac{1}{t'}$ per unit time. Combining these inequalities, we have $(p(S_k)-p(S_{k-1}))\frac{1}{t_k}\geq(1-\delta)opt(t',J\backslash S_{k-1})\frac{1}{t'}\geq(1-\delta)p(OPT^c,J\backslash S_{k-1})\frac{1}{U}$. This completes the proof. □

Based on the relation between $p(S_k)$ and $p(S_{k-1})$ in every two adjacent iterations in Lemma 1, we can establish the relation between $p(S_k)$ and $p(OPT^c)$ in the following lemma.

Lemma 2. *$p(S_k)\geq(1-\delta)(1-\Pi_{i=1}^{k}(1-\frac{t_i}{U}))p(OPT^c)$ for all $1\leq k\leq p$.*

Proof. Since $\frac{p(A_1)}{t_1}\geq(1-\delta)(\frac{p(OPT^c)}{U})$, it holds for the base case $k=1$, i.e., $p(A_1)\geq(1-\delta)\frac{t_1}{U}p(OPT^c)$. We prove the lemma by induction with the induction hypothesis $p(S_{k-1})\geq(1-\delta)(1-\Pi_{i=1}^{k-1}(1-\frac{t_i}{U}))p(OPT^c)$. With the hypothesis, we have

$$\begin{aligned}
p(S_k)&\geq p(S_{k-1})+(1-\delta)t_k\cdot\frac{p(OPT^c\backslash S_{k-1})}{U}\\
&\geq p(S_{k-1})+(1-\delta)\cdot\frac{t_k}{U}(p(OPT^c)-p(S_{k-1}))\\
&\geq(1-\delta)\left(1-\frac{t_k}{U}\right)p(S_{k-1})+(1-\delta)t_k\cdot\frac{p(OPT^c)}{U}\\
&\geq(1-\delta)\left(1-\frac{t_k}{U}\right)\left(1-\Pi_{i=1}^{k-1}\left(1-\frac{t_i}{U}\right)\right)p(OPT^c)\\
&\quad+(1-\delta)t_k\cdot\frac{p(OPT^c)}{U}\\
&=-(1-\delta)\Pi_{i=1}^{k}\left(1-\frac{t_i}{U}\right)p(OPT^c)\\
&\quad+(1-\delta)\left(1-\frac{t_k}{U}\right)p(OPT^c)+(1-\delta)t_k\cdot\frac{p(OPT^c)}{U}\\
&=-(1-\delta)\Pi_{i=1}^{k}\left(1-\frac{t_i}{U}\right)p(OPT^c)+(1-\delta)p(OPT^c)\\
&=(1-\delta)\left(1-\Pi_{i=1}^{k}\left(1-\frac{t_i}{U}\right)\right)p(OPT^c),
\end{aligned}$$

where the first inequality holds by Lemma 1, the second inequality follows by $p(OPT^c\backslash S_{k-1})\geq p(OPT^c)-p(S_{k-1})$, the fourth inequality holds by the induction hypothesis and the last three equalities follow by merging the items. □

Based on Lemmas 1 and 2, we derive the worst-case performance bound of the algorithm compared with the optimal profit in time-constrained CSC problem.

Lemma 3. *Algorithm BRP achieves within $\frac{2e}{(1-\delta)(e-1)}$ times of the optimal profit in time-constrained CSC problem with at most U units of power-on time.*

Proof. Assume that A_{p+1} is the last set which makes t_{p+1} exceed the power-on time constraints with $\sum_{i=1}^{p+1}t_i\geq U$. We have

$$\begin{aligned}
&p(S_p)+p(A_{p+1})\\
&\geq(1-\delta)\left(1-\Pi_{i=1}^{p+1}\left(1-\frac{t_i}{U}\right)\right)p(OPT^c)\\
&\geq(1-\delta)\left(1-\Pi_{i=1}^{p+1}\left(1-\frac{t_i}{\sum_{i=1}^{p+1}t_i}\right)\right)p(OPT^c)\\
&\geq(1-\delta)\left(1-\left(1-\frac{1}{p+1}\right)^{p+1}\right)p(OPT^c)\\
&\geq(1-\delta)\left(1-\frac{1}{e}\right)p(OPT^c),
\end{aligned}$$

where the first inequality follows by Lemma 2, the second inequality holds by the fact $\sum_{i=1}^{p+1}t_i\geq U$, the last two inequalities hold by the fact that the minimum value of $\Pi_{i=1}^{p+1}(1-\frac{t_i}{\sum_{i=1}^{p+1}t_i})$ is achieved with $t_1=t_2=\cdots=t_{p+1}$ and approaches $\frac{1}{e}$ with large $p+1$. Accordingly, $\max\{p(S_p),p(A_{p+1})\}\geq\frac{p(S_p)+p(A_{p+1})}{2}\geq\frac{(1-\delta)(e-1)}{2e}p(OPT^c)$. Therefore, the algorithm returns $\max\{p(S_p),p(\bar{S})\}\geq\max\{p(S_p),p(A_{p+1})\}\geq\frac{(1-\delta)(e-1)}{2e}p(OPT^c)$, thus is within $\frac{2e}{(1-\delta)(e-1)}$ times of the optimal solution in time-constrained CSC problem with at most U units of power-on time. □

Finally, we prove the feasibility and constant approximation of the proposed algorithm in the original CSC problem, as concluded in the following theorem.

Theorem 1. *Algorithm BRP achieves $\frac{2e}{(1-\delta)(e-1)\alpha}$-approximation for CSC problem, i.e., approximately within 4.5-6.4 times of the optimal solution with typical idle power $\alpha\in[0.5,0.7]$.*

Proof. Algorithm BRP schedules the VM execution with at most U units of power-on time, thus uses at most U units of (peak) power. Hence, it is feasible for the original CSC problem. Furthermore, it returns a profit that is at least $\frac{(1-\delta)(e-1)}{2e}p(OPT^c)$ where $p(OPT^c)$ is the optimal profit in time-constrained CSC problem with at most U units of power-on time. Let $p(OPT^{\hat{c}})$ be the optimal profit of time-constrained CSC problem with at most $\frac{U}{\alpha}$ units of power-on time. Obviously, $p(OPT^c)\geq\alpha p(OPT^{\hat{c}})$ since the number of power-on time units in $p(OPT^c)$ is α times that of $p(OPT^{\hat{c}})$. Moreover, it is easy to see that any feasible solution of the original CSC problem can utilize at most $\frac{U}{\alpha}$ units of power-on time according to the definition of idle power, thus is a feasible solution of the time-constrained CSC problem with $\frac{U}{\alpha}$ units of power-on time. Thus, we have $p(OPT^{\hat{c}})\geq p(OPT)$. Therefore, Algorithm BRP returns a profit that is at least $\frac{(1-\delta)(e-1)\alpha}{2e}p(OPT)$ and hence is $\frac{2e}{(1-\delta)(e-1)\alpha}$-approximation for the original CSC problem. □

5 Algorithm Design for General CSC Problem

In this section, we study the general case in which there exists a limitation on the number of PMs for re-provisioning. We try to develop a schedule by combining two algorithms to achieve a good average performance as well as constant approximation in terms of worst-case performance.

According to the overview of our idea described in Section 3, we can simply extend BRP to get the algorithm

BRP* for the general case, by iteratively activating new PMs until no more PMs are available. In general, this algorithm may perform well in average case, but a simple instance can be easily found to show that it has unbounded worst-case performance. Developing an algorithm with worst-case bounded performance for the general case is quite difficult. With the limit on the available PMs, packing VMs with similar deadlines to a PM, as BRP* does, may just choose VMs with short VMs and use up the PMs easily while achieving only a low total profit on each PM.

To address this challenge, we go another way around. Instead of directly addressing the original problem, we introduce a novel fractional version of the problem, called Fractional Time-constrained CSC (FCSC) problem, which has nice structural properties and can be solved optimally. Thus, we first develop a non-trivial algorithm to optimally solve it, and then transform the fractional solution back to be the (integral) solution of the original problem. Based on the optimality of the fractional solution returned for FCSC, we will ensure that the transformed solution would achieve a total profit close to the fractional solution, thereby achieving a low loss in approximation ratio.

The arrangement of this section is as follows. We first introduce the FCSC problem. Next, we develop an optimal algorithm for the fractional CSC problem. Then, we transform the solution of the fractional CSC problem back to be a feasible solution of the original problem and prove its constant approximation. Last, we combine the two algorithms developed to get the final service continuity strategy.

5.1 A Fractional Time-Constrained CSC Problem

In this section, we still define the time-constrained CSC problem to be the CSC problem with the power constraint replaced by the time-constraint that the total number of power-on time units used is at most U, and further introduce a fractional time-constrained CSC problem. We will prove that the optimal solution of the fractional time-constrained CSC problem can be computed and transformed to be a feasible solution of the CSC problem later.

Recall that a feasible solution of time-constrained CSC problem integrally selects the VMs and re-provisions/consolidates them to the PMs with the satisfaction of resource capacity constraints and power-on time constraints. We can treat each VM as two dimensional rectangle, with one dimension to be the size/height and the other dimension to be the length/width. Re-provisioning the selected VMs to a PM with capacity V is equivalent to placing the rectangles into a large rectangle with height V.

We first define the *virtual PMs* before introducing the fractional CSC problem. Take each PM as a virtual PM such that a VM can request just a part (but not all) of its demand from a virtual PM. Further consider that all m virtual PMs form a large virtual PM with capacity mV. With such an assumption, the assignment constraint is relaxed and a VM is allowed to be allocated to two virtual PMs. For example, when VM j is allocated to two virtual PMs 1 and 2 respectively with demand s_j^1 and s_j^2, then virtual PMs 1 and 2 respectively need to provide s_j^1 and s_j^2 units of resource (and thus keep operating) during the period $[0, d_j)$. Thus, we assume that, when hosting the VMs, the virtual PM needs to keep operating until the largest deadline of the VMs allocated even if it just serves partial resource demand of the VM. The total execution time of the virtual PMs still should not violate the power-on time constraints.

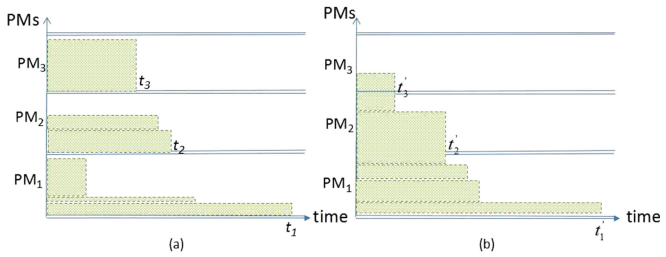


Fig. 2. An example showing how to transform the optimal solution of time-constrained CSC problem to be a feasible solution of FCSC problem without using more resources or power-on time.

With the assumptions of virtual PMs above, the *fractional time-constrained CSC problem* (FCSC problem) is defined as follows. It integrally selects a set of VMs and allocates them to the virtual PMs under both the resource capacity constraint on virtual PMs and the time-constraint that the total number of power-on time units used is at most U. The objective is to maximize the profit of VMs selected and allocated. Note that for FCSC problem, any set of VMs with total size not exceeding mV can be placed into the virtual PMs, without violating the resource capacity constraints since the VMs are allowed to be allocated to two virtual PMs. However, the power-on time-constraints on the virtual PMs may be violated.

An important property for the FCSC problem introduced above is that its optimal solution is at least that of the time-constrained CSC problem, thus can provide an upper bound for the original CSC problem. Its proof also indicates the existence of a novel algorithm for computing the optimal solution of the FCSC problem.

To prove that the optimal profit of FCSC problem is at least that of the time-constrained CSC problem, it is sufficient to transform the optimal solution of the time-constrained CSC problem to be a feasible solution of the FCSC problem. That is, we show that all the VMs selected in the optimal solution of time-constrained CSC problem can be allocated to virtual PMs and get a feasible solution for FCSC problem. Assume that J_{opt} is the set of VMs selected in the optimal solution of time-constrained CSC problem. We allocate the VMs in J_{opt} to the virtual PMs in the order of non-increasing length as follows. We allocate the first VM with the largest length to the first virtual PM, and then greedily allocate the next VM and activate a new virtual PM only when the resources on the current virtual PM are uses up. Fig. 2 demonstrates an example showing the transformation.

The resulting allocation can be proved to be a feasible solution for FCSC problem with the assumption that a VM is allowed to be allocated to two virtual PMs, which is concluded in the following lemma.

Lemma 4. *The optimal profit for FCSC problem is at least that of the time-constrained CSC problem.*

Proof. Let J_{opt} be the optimal solution of CSC problem. Note that the transformation takes J_{opt} as a input. The VMs in J_{opt} are sorted in the order of non-increasing length and allocated one by one to the virtual PMs. The proof is simply based on the observation that such an

allocation always activates a new virtual PM when the resources on activated ones are fully occupied and the new virtual PM is activated using the minimum length of power-on time. Such a transformed allocation is obviously a feasible solution for FCSC problem and does not utilize more resources or power-on time than J_{opt} of CSC problem. This completes the proof. □

Consider the large virtual PM with mV units of resource that is formed by all virtual PMs to serve the VMs. Take as if each unit of the resource is taken from a space with addresses in $[1, mV]$. If the unit of resource in address $c \in [1, mV]$ is allocated to VM j, we say VM j *occupies* address c.

The transformation rule defined for proving Lemma 4 implies the following critical property of the optimal solution for FCSC problem, which will help us to design its optimal algorithm. That is, it makes the activated virtual PMs, except the last one, use up their resource. Thus, equivalently, we can say that VMs in J_{opt} *occupy* all the addresses in $[1, s_{opt}]$ or addresses in $[1, s_{opt}]$ are *fully occupied* by VMs in J_{opt}. In general, there is an immediate lemma for the optimal solution of FCSC problem following by Lemma 4.

Lemma 5. *Assuming that J_{fopt} is the set of VMs chosen by the optimal solution of FCSC problem, then there exists an optimal allocation for FCSC problem where VMs in J_{fopt} occupy all the addresses in $[1, s_{fopt}]$ with $s_{fopt} = \sum_{j \in J_{fopt}} s_j$, and moreover, a VM with larger length occupies the lower address.*

5.2 Optimal Algorithm for FCSC Problem

Based on the definition of FCSC problem and its properties introduced above, we develop a novel algorithm to optimally solve the FCSC problem. To find the optimal solution for FCSC problem, we still have to deal with the power-on time-constraints and PM constraint when VMs are allocated to virtual PMs.

The key idea is to develop a dynamic programming algorithm by seeking a recursion function (or recursive sub-problem) for FCSC problem. One possible idea, which is intuitive but may fail, is that we can try to find the maximum achievable profit using the addresses in $[1, c]$ and find the largest one among all possible c with $1 \le c \le \min\{mV, s_{sum}\}$ to find the optimal profit. With such an intuition, the method to compute the optimal profit for allocating n VMs is to find the larger value between the profit found in two recursive sub-problems, the maximum profit for allocating the first $n-1$ VMs in addresses $[1, c]$ when the nth VM is not allocated and the maximum profit for allocating the first $n-1$ VMs in addresses $[1, c - s_n]$ when the nth VM is allocated. However, such an intuition fails in dealing with the power-on time constraints when designing the recursion function. This is because, if VM j is allocated to a virtual PM that is not activated before allocation, then it needs d_j more power-on time; if VM j is allocated to a virtual PM that is already activated, then it may need no more power-on time. The problem above lies in the following fact: a recursion function for dynamic programming function needs to compute the function in a bottom-up manner; but when comparing these two sub-problems, we have no information on whether the PM before allocation is activated or not.

We note that the above intuition fails because when we consider the allocation for the current VM, we are lacking of the structure information for the recursive sub-problem before allocating the current VM.

Thus, we try to make more structural information for the problem. Note that there are multiple optimal allocations for FCSC problem that can achieve the same optimal profit. Recall that Lemma 5 implies that there exists an optimal solution for FCSC problem where the selected VMs fully occupy the addresses starting from address 1 and ending at address s_{fopt}, and furthermore, a VM with larger length is allocated first to lower address.

We utilize such a key property to design the recursion function. With such a property, we just try to find the optimal allocation which fully occupies the addresses in $[1, s_{fopt}]$ but ignore all other optimal allocations, and further resort the VMs so that $s_1 \ge s_2 \ge \cdots \ge s_n$. Such an idea restricts our search space, but allows us to assume that the virtual PM, whose resources are partially used but not used up before allocating the current VM, is already activated.

Thus, given an address space $[1, c]$, our objective is reduced to computing the maximum profit by selecting a subset J_c of VMs in the first k VMs so that the total size $\sum_{j \in J_c, j \le k} s_j = c$ (which implies that all the VMs in J_c fully occupy the addresses in $[1, c]$ and meanwhile the last selected VM is ending at address c), and meanwhile, the virtual PMs for allocating such VMs use at most t units of power-on time. Let function $p(k, c, t)$ return such a maximum profit.

The definition above helps make the recursive sub-problem well-structured, which is able to lead to a novel recursive function below to compute $p(k, c, t)$. A tricky part of the proof for deriving this recursion function is how to identify the case that the addresses in $[1, c]$ are not possible to be fully occupied by integrally selecting from the first k VMs.

Lemma 6. *The recursion function to compute $p(k, c, t)$ is as follows,*

$$p(k,c,t) = \begin{cases} \max \begin{cases} p(k-1,c,t) \\ p(k-1,c-s_k,t), \\ \qquad if\ c - s_k > V \cdot (\lceil \frac{c}{V} \rceil - 1) \\ p(k-1,c-s_k,t-d_k), \\ \qquad if\ 0 \le c - s_k \le V \cdot (\lceil \frac{c}{V} \rceil - 1) \end{cases} \\ 0, \quad if\ c = 0\ and\ t \ge 0 \\ -\infty, \quad if\ j = 0\ and\ c > 0 \\ -\infty, \quad if\ c - s_k < 0 \\ -\infty, \quad if\ t < d_k \end{cases} \tag{11}$$

Proof. Recall the definition of $p(k, c, t)$. If $p(k, c, t)$ is the optimal profit of FCSC problem, then this implies that $c = s_{jopt}$ and the chosen VMs fully occupy the addresses in $[1, c]$ and the lower address is occupied by the VM with larger length.

According to the definition of the recursion function, if $p(k, c, t)$ is the optimal solution, then this implies that $c = s_{opt}$ and all addresses in $[1, c - s_k]$ are occupied before allocating VM k. Thus, we can simply assume that all addresses in $[1, c - s_k]$ are already occupied and the first $\lceil \frac{c - s_k}{V} \rceil$ virtual PMs that contribute at least one resource in addresses $[1, c - s_k]$ is already activated.

With such a structural information, we know VM k should occupy addresses in $[c-s_k+1, c]$ that is belonging to the $\lceil \frac{c}{V} \rceil$-th virtual PM. If $\lceil \frac{c}{V} \rceil - \lceil \frac{c-s_k}{V} \rceil = 0$ or equivalently $c - s_k > V \cdot (\lceil \frac{c}{V} \rceil - 1)$, then the $\lceil \frac{c}{V} \rceil$-th virtual PM already hosts at least one VM among the first $k-1$ VMs. This implies that it is already activated for execution with larger than d_k time units before allocating VM k. Thus, no more power-on time need to be used to activate the $\lceil \frac{c}{V} \rceil$-th virtual PM when allocating VM k. If $\lceil \frac{c}{V} \rceil - \lceil \frac{c-s_k}{V} \rceil = 1$, then the $\lceil \frac{c}{V} \rceil$-th virtual PM does not host any VM before allocating VM k and d_k more units of power-on time should be used to activate it. Consequently, when VM k is selected, we can set $p(k,c,t) = p(k-1, c-s_k, t) + p_k$ if $c - s_k > V \cdot (\lceil \frac{c}{V} \rceil - 1)$ and $p(k,c,t) = p(k-1, c-s_k, t-d_k) + p_k$ if $0 \le c - s_k \le V \cdot (\lceil \frac{c}{V} \rceil - 1)$. If VM k is not selected, then we have $p(k,c,t) = p(k-1,c,t)$. Obviously, we need to set $p(\cdot, 0, t) = 0$ if $c = 0$ and $t \ge 0$ for the initialization. The analysis above has specified the main idea of the recursion function, which relies on the assumption that the address in $[1, c]$ are fully occupied.

However, one more critical issue has not been addressed, which is also a tricky part in the design. Since the dynamic programming algorithm needs to compute a general $p(k,c,t)$ in a bottom-up manner, we need to tackle the case that the addresses $[1, c]$ are not possible to be fully occupied by selecting from the first k VMs.

To identify this situation, we set $p(k,c,t) = -\infty$ when using the first k VMs is impossible to fully utilize the resources in $[1, c]$. Thus, we need to identify the condition with which $p(k,c,t)$ is set to be $-\infty$. In general, the last selected VM will occupied the address c according to the definition of $p(k,c,t)$. We discuss all possible cases of the recursion function. If $t < d_k$, then this implies any VM among the first k VMs has a length larger than t, thus it is impossible to fully utilize the addresses in $[1, c]$ when $c \neq 0$ and hence $p(k,c,t) = -\infty$. If $c - s_k < 0$ then k should not be allocated and $p(k,c,t) = p(k-1,c,t)$. If $j = 0$ and $c > 0$, then obviously $p(k,c,t) = -\infty$. This completes the proof in deriving the recursive function. □

Recall that we want to find the optimal allocation that the selected VMs fully occupy the addresses in $[1, c]$. Accordingly, the defined function $p(k,c,t)$ returns a positive value only when the addresses in $[1, c]$ are fully occupied by the selected VMs.

Finally, to find the optimal profit for FCSC problem, denoted by $OPT(FCSC)$, we need to enumerate $c \in [1, \min\{mV, s_{sum}\}]$ to find exactly the value $c = s_{fopt}$. Thus, the optimal profit for FCSC problem can be computed by,

$$OPT(FCSC) = \max_{1 \le c \le \min\{mV, s_{sum}\}} p(n, c, U). \quad (12)$$

More details can be found in Algorithm FRACTIONALCSC which implements the dynamic programming method stated above. The following theorem concludes the optimality of FRACTIONALCSC.

Theorem 2. *Algorithm FRACTIONALCSC optimally solves FCSC problem in $O(m^2nUV^2)$ steps.*

Proof. The optimality of FRACTIONALCSC follows directly from the recursion function we derived.

To calculate the recursion function $p(k,c,t)$, we need to enumerate n possible values of the first parameter, at most mV values of the second parameter and U possible values of the last one. In the final step, to find the value $\max\{p(n,c,U)\}$, another mV possible values should be enumerated. Therefore, the time complexity is $O(m^2nUV^2)$. □

Algorithm 2. FRACTIONALCSC(J, U, m)

```
1: for c = 1 to min{mV, s_sum} do
2:   Set p(0, ·, ·) = −∞.
3:   Set p(·, 0, t) = 0 for all t ≥ 0.
4:   for k = 1 to n do
5:     for cc' = 1 to c do
6:       for t = 1 to U do
7:         if t < d_k or cc' < 0 then
8:           p(k, cc', t) = −∞
9:         else
10:          temp1 = p(k − 1, cc', t)
11:          if cc' − s_k > V · (⌈cc'/V⌉ − 1) then
12:            temp2 = p(k − 1, cc' − s_k, t) + p_k
13:          else if 0 ≤ cc' − s_k ≤ V · (⌈cc'/V⌉ − 1) then
14:            temp2 = p(k − 1, cc' − s_k, t − d_k) + p_k
15:          else
16:            p(k, cc', t) = −∞
17:          end if
18:          p(k, cc', t) = max{temp1, temp2, temp3}
19:        end if
20:      end for
21:    end for
22:  end for
23:  retP = max{p(n, c, U)}
24:  Find the set of selected VMs S in the recursion function
     with a backward search.
25: end for
26: return (retP, S)
```

5.3 Transforming from Fractional Solution to Integral Solution

Now we transform the optimal solution of FCSC problem to be a feasible solution of the CSC problem without losing much performance. Recall that in FCSC problem, it allows the VMs to be divisible when they are assigned to PMs. Therefore, we need to transform the fractional solution to be the (integral) solution of CSC problem.

The main difficulty lies in the fact that the transformation may result in the utilization of more than m PMs and more than U units of power-on time. The transformation idea is based on the fact that the optimal allocation for FCSC problem fully occupies the addresses in $[1, s_{fopt}]$, as stated in Lemma 4, which can be used to control the usage in the number of PMs that is needed for transformation.

Thus, we use m PMs and U units of power-on time to recover the fraction solutions that uses $\lfloor \frac{m}{2} \rfloor$ PMs and $\lfloor \frac{U}{2} \rfloor$ power-on time. For the ease of discussion, we only discuss the case that $\frac{U}{2}, \frac{m}{2}$ are integers since the flooring operation does not affect the constant approximation of the algorithm.

Let $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ be the optimal solution of FCSC problem with the input of $\frac{U}{2}$ power-on time and $\frac{m}{2}$ PMs. According to Lemma 5, all VMs fractionally assigned to PMs in $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ are allocated to at most two

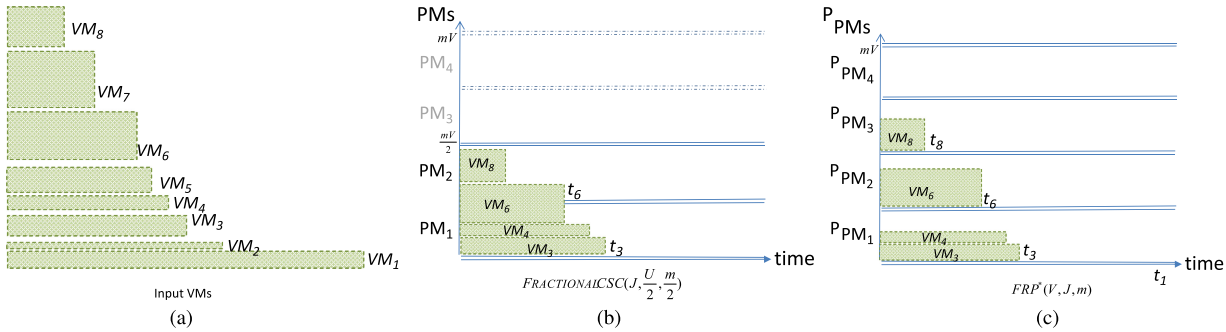


Fig. 3. An example showing the results returned by Algorithm FRP* where U is set with $U = 2(d_3 + d_6)$ and the profit of each VM/rectangle is equal to the size of its area. (a) All VMs sorted by length, as the input of the dynamical programming algorithm FractionalCSC. (b) A solution returned by Algorithm FractionalCSC($J, \frac{U}{2}, \frac{m}{2}$) using $\frac{U}{2}$ units of power-on time and $\frac{m}{2}$ PMs, which is the optimal solution of the fractional time-constrained CSC problem, $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$. (c) Solution returned by FRP*(V, J, m), which is a feasible solution of CSC problem transformed from $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ using at most U units of power-on time and m PMs.

PMs. Thus, we can assign the PMs in $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ one by one in the order of non-increasing length and activate at most twice of the PMs to integrally assign all these VMs. More details can be found in Algorithm FRP*.

In terms of the time complexity of Algorithm FRP*, computing the optimal solution $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ would cost $O(m^2nUV^2)$ time when calling Algorithm FractionalCSC, while allocating the VMs chosen by FractionalCSC would cost at most $O(nm)$ time. Thus, the running time of Algorithm FRP* is $O(m^2nUV^2)$. Fig. 3 demonstrates an exemplary solution returned by Algorithm FRP*.

According to Lemma 4, the algorithm for FCSC problem finds the optimal allocation that fully occupies addresses in $[1, s_{fopt}]$ and a VM with larger length occupies lower address. Thus, activating twice the number of PMs and power-on time used in FractionalCSC($J, \frac{U}{2}, \frac{m}{2}$) is enough to get an integral solution, which satisfies the PM constraints and power-on time constraints in the time-constrained CSC problem. Hence, it further satisfies the power constraint of the original CSC problem since using U units of power-on time implies that the total power used is at most U units of (peak) power. Therefore, the algorithm outputs a feasible solution for CSC problem.

Algorithm 3. FRP*(V, J, m)

1: $(p, S) =$ FractionalCSC($J, \frac{U}{2}, \frac{m}{2}$)
2: sort VMs in S by their length in a non-decreasing order
3: **for all** $\hat{j}$ in S **do**
4: **if** VM $\hat{j}$ is fractionally selected **then**
5: let VM j be the VM with full demand corresponding to $\hat{j}$.
6: **else**
7: $j = \hat{j}$
8: **end if**
9: **if** VM j can be put into the current PM **then**
10: put VM j into the current PM
11: **else**
12: activate a new PM and put VM j into the new PM
13: **end if**
14: $profit = profit + p_j$
15: **end for**
16: **return** $profit$

Based on the the discussion above, we further prove the constant approximation of Algorithm FRP* by applying the upper bound established in Lemma 4.

Theorem 3. *Algorithm FRP* is $\frac{4}{\alpha}$-approximation for CSC problem, i.e., approximately within 5.7-8 times of the optimal solution with typical idle power $\alpha \in [0.5, 0.7]$.*

Proof. The profit achieved in Algorithm FRP* is no less than that of $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ since all VMs selected in $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$ are executed. That is, FRP*$(J, U, m) \geq OPT(FCSC, J, \frac{U}{2}, \frac{m}{2})$. Obviously, we have $OPT(FCSC, J, \frac{U}{2}, \frac{m}{2}) \geq \frac{1}{4}OPT(FCSC, J, U, m)$ because the input VMs are the same and the input parameters are scaled by a factor of 2. Moreover, assuming that $OPT(CSC^c, J, U, m)$ is the optimal solution of the time-constrained CSC problem, then $OPT(FCSC, J, U, m) \geq OPT(CSC^c, J, U, m)$ according to Lemma 4. Therefore, it is true that FRP*$(J, U, m) \geq OPT(FCSC, J, \frac{U}{2}, \frac{m}{2}) \geq \frac{1}{4}OPT(FCSC, J, U, m) \geq \frac{1}{4}OPT(CSC^c, J, U, m)$. Let $OPT(CSC)$ be the optimal profit of CSC problem. Applying similar proof in Theorem 1, we can further derive that $OPT(CSC^c, J, U, m) \geq \alpha OPT(CSC)$. Thus, FRP*$(J, U, m) \geq \frac{\alpha}{4}OPT(CSC)$ and FRP* is $\frac{4}{\alpha}$-approximation. □

5.4 Combining the Results

As what is shown above, Algorithm FRP* has a constant bound in terms of worst-case performance. In order to further enhance its practical significance, we combine it with BRP* that has a good average performance in general, so as to achieve both good average performance and theoretically worst-case bounded performance. The method is to simply run these two algorithms once and return the one that achieves higher profit. Algorithm CSC-Schedule presents the final algorithm.

Algorithm 4. CSC-Schedule(V, J, m)

1: $profit_1$ = BRP*(V, J, m)
2: $profit_2$ = FRP*(V, J, m)
3: **return** $\max\{profit_1, profit_2\}$

Obviously, CSC-Schedule has the same constant approximation ratio as FRP*, thus its worst-case performance is well-bounded.

6 Simulation Results

The theoretical analysis has verified the worst-case performance bounds of the algorithm proposed in this paper. In this

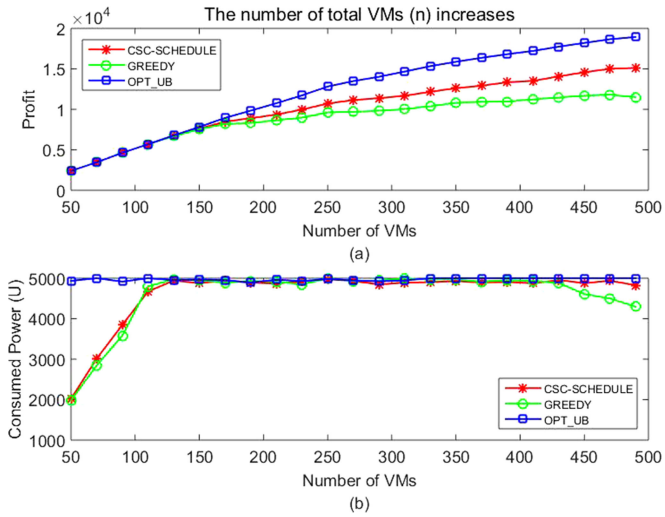


Fig. 4. (a) Profit achieved when the number of VMs varies. (b) Amount of energy consumed when the number of VMs varies.

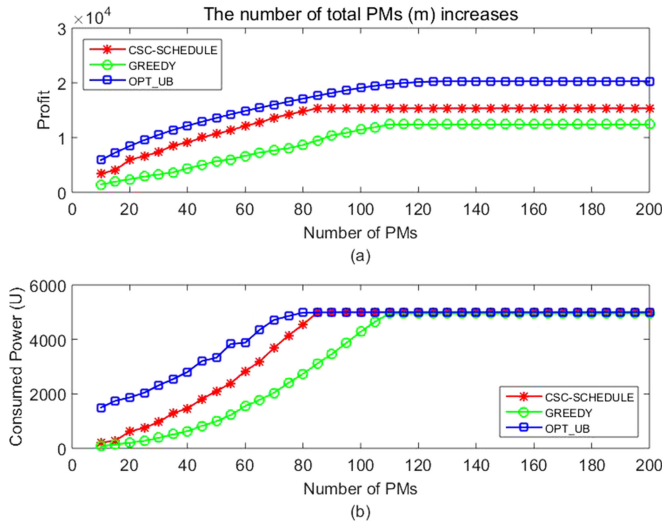


Fig. 5. (a) Profit achieved when the number of PMs varies. (b) Amount of energy consumed when the number of PMs varies.

section, we perform simulations for the algorithm to further validate its average performances over the achievable profit.

No prior works have addressed the service continuity problem studied in this paper, thus we compare our algorithm with the natural greedy schedule mentioned in Section 4. Furthermore, we compare the performance of our algorithm with the optimal solution. Since computing the optimal solution for the CSC problem is NP-hard, we compare the performance of CSC-SCHEDULE with the upper bound of the optimal solution, denoted as *OPT_UB*, which is obtained by relaxing the ILP formulation to be LP formulation with $x_{ij} \in [0, 1]$. Since no real-trace has provided a dataset similar to that of the continuity maintenance problem with VM requirements considered in this paper, we consider random input of VMs in our simulations.

In the simulation, we set the power-restored time $T = 100$(min). We assume that there are $m = 100$ homogeneous PMs, each with 32 CPU cores. Each PM with full utilization would consume one (normalized) unit of energy per minute. We set the power budget to be $U = 5{,}000$. The deadline of each VM is assumed to be a random number in $[1, T]$. The demand of each VM (number of CPU cores requested) is assumed to be an integer randomly ranging in $[1, 32]$. The profit of the VMs is uniformly generated from $[1, 100]$. The parameters of the power consumption function are set with $\alpha = 0.5, \mu = 1$. a) Impact of the Number of VMs. Fig. 4a shows the change in the profit achieved by different schedules when the number of VMs increases in the range of $[50, 500]$ with a step of 20. The curves are generated by increasing the number of VMs, where each point on the curves of the results is generated in a single run. Generally, the profit of the algorithms increases with the number of VMs. The profit achieved by our algorithm CSC-SCHEDULE is much higher (up to 25 percent) than the greedy schedule. Moreover, we can see from the figure that the profit achieved approaches that of *OPT_UB* (within 80 percent), thus is even closer to the optimal solution.

Fig. 4b illustrates the amount of power used by different schedules when the number of VMs varies. At the beginning, the power used by CSC-SCHEDULE (and Greedy) increases with the rise of the number of VMs and then reaches the limit $U = 5{,}000$ with $n \geq 150$, where U becomes the bottleneck, while *OPT_UB* almost always uses up the energy since it can produce fractional (but maybe infeasible) solutions. We can see from Figs. 4a and 4b that the profit achieved by the three schedules are nearly the same before the available power becomes the bottleneck, and after that, the profit achieved differs much for the three schedules. 2) Impact of the Number of PMs. Fig. 5a shows the profit achieved by the three schedules when the number of PMs varies. In this scenario, the number of PMs $m \in [10, 200]$, the number of VMs $n = 500$, and the amount of power-on time $U = 5{,}000$. The profit obtained by CSC-SCHEDULE is about 20-40 percent higher than that of the greedy schedule. Moreover, it achieves a profit close to (within 75 percent) that of *OPT_UB*, thus even approaches that of the optimal solution. The profit of the algorithms increases with the number of PMs and becomes stable when $m \geq 110$.

Fig. 5b shows the amount of power consumed by different schedules when the number of PMs varies. The trends of lines in this figure are similar. The amount of power used increases when $m \leq 100$ and becomes stable after that because all schedules almost use up the energy. We can see from Figs. 5a and 5b that the profit achieved by CSC-SCHEDULE and Greedy becomes stable when the available power is used up.

The simulations above have demonstrated the good average performance on maximizing the profit of our proposed algorithm. In Table 1, we further compare the time complexity of the algorithms. Although the greedy algorithm runs fastest (with $O(n \cdot \max\{\log n, m\})$ time) due to its greedy nature, it achieves 25 percent less profit in providing service continuity than CSC-SCHEDULE does, as demonstrated in the simulation results above. The running time of CSC-SCHEDULE depends on the maximum one between BRP* and FRP*.

TABLE 1
Time Complexities of the Algorithms

BRP*	FRP*	CSC-SCHEDULE	Greedy
$O(mnVT)$	$O(m^2nV^2U)$	$O(mnV \cdot \max\{T, mVU\})$	$O(n \cdot \max\{\log n, m\})$

Algorithm BRP* runs in $O(mnVT)$ time, while Algorithm FRP* runs in $O(m^2nV^2U)$ time. In our simulation, we found that around 98 percent of the solutions returned by Algorithm BRP* have higher profit that of FRP*. Thus, in practice, we can just return the solution of BRP* as the final solution of CSC-SCHEDULE to significantly reduce its running time to be $O(mnVT)$ without losing much profit.

Combining with the theoretical bound derived for the worst-case performance, these together verify the efficiency of our proposed algorithm.

7 Future Works and Concluding Remarks

In this paper, we introduce and theoretically study the scheduling problem to maintain cloud service continuity with maximum profit under power shortage. We develop efficient scheduling algorithms with theoretical guarantees/small approximation ratios to maximize the profit of VMs of which the service continuity requirements are satisfied.

This work has conducted a theoretical study on how to provide service continuity under power outage. Our theoretical study can shed some light on the service continuity strategy design under power shortage. In our preliminary study, we focus on the resource of CPUs and vCPUs considering that the CPU usage takes up significant share of power needed. In future works, it is worth extending the study to the case with multiple types of resources. Although the basic algorithmic idea introduced in this paper can be extended to be adaptive to the multi-resource scenario, however, it is quite challenging to develop algorithms with theoretical performance guarantee since there exist multiple capacity constraints with respect to the multiple types of resources. Thus, we would like to leave it as an open problem and study it in future work.

Acknowledgments

The work is supported in part by Hong Kong Research Grant Council under GRF 120612 and CRF C7036-15G, GRF 11213114, NSFC-Guangdong Joint Fund under project U1501254, National Natural Science Foundation of China under Grants No. 61672154, No. 61632008, No. 61320106007, No. 61572310, No. 61672370, Jiangsu Provincial Key Laboratory of Network and Information Security under Grant No. BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No. 93K-9, Collaborative Innovation Center of Wireless Communications Technology and Collaborative Innovation Center of Social Safety Science and Technology.

References

[1] Q. P. S. Team, "Average cost of data center outages: $627,418 per incident," Quality Power Solutions Ltd, Mar. 18. 2015. [Online]. Available: http://www.qpsolutions.net/2015/03/average-cost-of-data-center-outages-627418-per-incident/

[2] Data center knowledge, Penton Ltd. [Online]. Available: http://www.datacenterknowledge.com/archives/category/manage/uptime/

[3] G. Smith, "Amazon power outage exposes risks of cloud computing," 2012. [Online]. Available: http://www.huffingtonpost.com/2012/07/02/amazon-power-outage-cloud-computing_n_1642700.html

[4] Y. Qu, G. Yang, and X. Liu, "Aliyun data center in HK suffers 14-hour disruption," 2015. [Online]. Available: http://english.caixin.com/2015-06-24/100822037.html

[5] S.-Y. Jing, S. Ali, K. She, and Y. Zhong, "State-of-the-art research study for green cloud computing," *J. Supercomputing*, vol. 65, no. 1, pp. 445–468, 2013.

[6] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proc. IEEE*, vol. 102, no. 1, pp. 11–31, Jan. 2014.

[7] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1378–1391, Oct. 2013.

[8] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proc. IEEE INFOCOM*, 2011, pp. 1332–1340.

[9] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "EnaCloud: An energy-saving application live placement approach for cloud computing environments," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2009, pp. 17–24.

[10] M. A. Adnan, R. Sugihara, and R. K. Gupta, "Energy efficient geographical load balancing via dynamic deferral of workload," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 188–195.

[11] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 123–134, 2009.

[12] Z. Liu, et al., "Renewable and cooling aware workload management for sustainable data centers," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 40, no. 1, pp. 175–186, 2012.

[13] S. Ren, Y. He, and F. Xu, "Provably-efficient job scheduling for energy and fairness in geographically distributed data centers," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 22–31.

[14] S.-Y. Jing, S. Ali, K. She, and Y. Zhong, "State-of-the-art research study for green cloud computing," *J. Supercomputing*, vol. 65, no. 1, pp. 445–468, 2013.

[15] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating server idle power," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 205–216, 2009.

[16] C.-C. Lin, P. Liu, and J.-J. Wu, "Energy-efficient virtual machine provision algorithms for cloud systems," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, 2011, pp. 81–88.

[17] R. Urgaonkar, B. Urgaonkar, M. J. Neely, and A. Sivasubramaniam, "Optimal power cost management using stored energy in data centers," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 2011, pp. 221–232.

[18] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. K. Fathy, "Energy storage in datacenters: What, where and how much?" in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 2012, pp. 187–198.

[19] H. Xu and B. Li, "Reducing electricity demand charge for data centers with partial execution," in *Proc. Int. Conf. Future Energy Syst.*, 2014, pp. 51–61.

[20] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew, "Greening geographical load balancing," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 2011, pp. 233–244.

[21] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2011, Art. no. 59.

[22] H. Xu, C. Feng, and B. Li, "Temperature aware workload management in geo-distributed datacenters," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 41, no. 1, pp. 373–374, 2013.

[23] Z. Xu and W. Liang, "Minimizing the operational cost of data centers via geographical electricity price diversity," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 99–106.

[24] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 123–134, 2009.

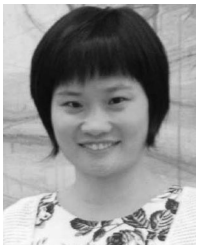
[25] T. Wood, E. Cecchet, K. Ramakrishnan, P. Shenoy, J. Van Der Merwe, and A. Venkataramani, "Disaster recovery as a cloud service: Economic benefits & deployment challenges," in *Proc. 2nd USENIX Workshop Hot Topics Cloud Comput.*, 2010, pp. 1–7.

[26] M. Klems, S. Tai, L. Shwartz, and G. Grabarnik, "Automating the delivery of it service continuity management through cloud service orchestration," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2010, pp. 65–72.

[27] C. Develder, J. Buysse, B. Dhoedt, and B. Jaumard, "Joint dimensioning of server and network infrastructure for resilient optical grids/clouds," *IEEE/ACM Trans. Netw.*, vol. 22, no. 5, pp. 1591–1606, Oct. 2014.

[28] M. F. Habib, M. Tornatore, M. De Leenheer, F. Dikbiyik, and B. Mukherjee, "A disaster-resilient multi-content optical datacenter network architecture," in *Proc. 13th Int. Conf. Transparent Opt. Netw.*, 2011, pp. 1–4.
[29] G. Chen, et al., "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. 5th USENIX Symp. Netw. Syst. Des. Implementation*, 2008, vol. 8, pp. 337–350.
[30] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 13–23, 2007.
[31] W. Dargie, "Estimation of the cost of VM migration," in *Proc. 23rd Int. Conf. Comput. Commun. Netw.*, 2014, pp. 1–8.
[32] E. L. Lawler, "Fast approximation algorithms for Knapsack problems," *Mathematics Operations Res.*, vol. 4, no. 4, pp. 339–356, 1979.

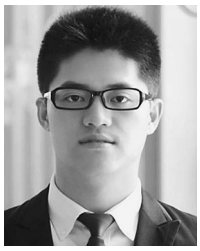
Weiwei Wu received the BSc degree from the South China University of Technology and the PhD degree from City University of Hong Kong (CityU, Department of Computer Science) and University of Science and Technology of China (USTC), in 2011, and went to Nanyang Technological University (NTU, Mathematical Division, Singapore) for post-doctorial research, in 2012. He is an associate professor with Southeast University, P.R. China. His research interests include optimizations and algorithm analysis, wireless communications, crowdsourcing, cloud computing, reinforcement learning, game theory, and network economics. He is a member of the IEEE.



Jianping Wang received the BS and MS degrees in computer science from Nankai University, Tianjin, China, in 1996 and 1999, respectively, and the PhD degree in computer science from the University of Texas at Dallas, in 2003. She is an associate professor in the Department of Computer Science at City University of Hong Kong. Her research interests include dependable networking, optical networks, cloud computing, service oriented networking, and data center networks. She is a member of the IEEE.

Kejie Lu received the BSc and MSc degrees in telecommunications engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively, and the PhD degree in electrical engineering from the University of Texas at Dallas, in 2003. In 2004 and 2005, he was a postdoctoral research associate in the Department of Electrical and Computer Engineering, University of Florida. In July 2005, he joined the Department of Electrical and Computer Engineering, University of Puerto Rico at Mayaguez, where he is currently an associate professor. His research interests include architecture and protocols design for computer and communication networks, performance analysis, network security, and wireless communications. He is a member of the IEEE.

Wen Qi received the BE degree from the Department of Automation, Nankai University, in 2009, the MS degree in computer science from the City University of Hong Kong, in 2013, and the PhD degree from the Department of Computer Science, City University of Hong Kong, in 2016. His research interests include cloud computing, networking, and security.



Feng Shan received the PhD degree in computer science from Southeast University, China, in 2015. He is currently an assistant professor in the School of Computer Science and Engineering, Southeast University. He was a visiting scholar in the School of Computing and Engineering, University of Missouri-Kansas City, Kansas City, Missouri, from 2010 to 2012. His research interests include the areas of energy harvesting, wireless power transfer, algorithm design and analysis. He is a member of the IEEE.

Junzhou Luo received the BS degree in applied mathematics and the MS and PhD degrees in computer network, all from Southeast University, China, in 1982, 1992, and 2000, respectively. He is a full professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He is a co-chair of IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design, and he is a member of the ACM and chair of ACM SIGCOMM China. His research interests include next generation network architecture, network security, cloud computing, and wireless LAN. He is a member of the IEEE Computer Society.

Research on the Architecture and its Implementation for Instrumentation and Measurement Cloud

Hengjing He, Wei Zhao, Songling Huang, Geoffrey C. Fox, and Qing Wang, *Senior Member, IEEE*

Abstract—Cloud computing has brought a new method of resource utilization and management. Nowadays some researchers are working on cloud-based instrumentation and measurement systems designated as Instrumentation and Measurement Clouds (IMCs). However, until now, no standard definition or detailed architecture with an implemented system for IMC has been presented. This paper adopts the philosophy of cloud computing and brings forward a relatively standard definition and a novel architecture for IMC. The architecture inherits many key features of cloud computing, such as service provision on demand, scalability and so on, for remote Instrumentation and Measurement (IM) resource utilization and management. In the architecture, instruments and sensors are virtualized into abstracted resources, and commonly used IM functions are wrapped into services. Users can use these resources and services on demand remotely. Platforms implemented under such architecture can reduce the investment for building IM systems greatly, enable remote sharing of IM resources, increase utilization efficiency of various resources, and facilitate processing and analyzing of Big Data from instruments and sensors. Practical systems with a typical application are implemented upon the architecture. Results demonstrate that the novel IMC architecture can provide a new effective and efficient framework for establishing IM systems.

Index Terms—Architecture, cloud computing, distributed computing, instrumentation and measurement cloud, parallel processing, power system state estimation, cloud service

1 Introduction

Since instrumentation and measurement (IM) technology is closely combined with information technology, development in information technology (IT) can lead to the advance of IM technology. In the early stages, computers were used to control instruments and sensors for local data acquisition and analysis. Later on, virtual instrumentation technology was developed and many of the functions that were implemented by hardware in instruments can now be achieved by computer software. With the development of networks and the internet, remote instrumentation and measurement (RIM) emerged as a new technology in IM field [1]. Such RIM technology has brought many benefits to related areas, especially to those areas that involve large distributed systems [2]. It can greatly facilitate instrument control and data acquisition and processing. Additionally, new computing paradigms, such as grid computing, can be integrated into IM technology to further improve the ability for data processing, and resource sharing and management of distributed IM systems [3]. Grid-enabled instrumentation and measurement system (GEIS) is a typical type of those systems. GEIS brings many advantages to data intensive IM applications and heterogeneous IM resource management. However, due to some limitations of grid computing, IM systems that integrate a grid are eventually not widely adopted in practical use.

Currently, most IM systems are built upon local architecture or traditional client/server (C/S) architecture as explained in [4]. In a local IM architecture, such as the main architecture of the NI LabVIEW platform, instruments and sensors are connected directly to the computer and it is difficult to build large IM systems. As for C/S architecture, instruments and sensors simply provide remote access interfaces through gateway servers to clients. However, both architectures require the user to build the entire IT system and to maintain all resources. Thus, there has to be a great investment in building the whole IM system and, besides, system stability, scalability and fault tolerance can be serious problems for both architectures. Moreover, to satisfy resource requirement for peak and valley load, the IM system should be built according to the peak load at the very beginning and, even if the load drops, the system cannot scale down accordingly. Therefore, the utilization rate of resource in the IM system can be quite low, especially for those systems with great load dynamics. In addition to the problems mentioned above, large amounts of data collected from various devices need much more powerful computing

- *H. He, W. Zhao, and S. Huang are with the State Key Lab. of Power System, Department of Electrical Engineering, Tsinghua University, Beijing 100084, China. E-mail: hehj11@mails.tsinghua.edu.cn, zhaowei@mail.tsinghua.edu.cn, huangsling@tsinghua.edu.cn.*
- *G.C. Fox is with the School of Informatics and Computing and CGL, Indiana University, Bloomington, IN 47405-7000. E-mail: gcf@indiana.edu.*
- *Q. Wang is with the School of Engineering and Computing Sciences, Durham University, Durham DH1, United Kingdom. E-mail: qing.wang@durham.ac.uk.*

Manuscript received 23 May 2016; revised 16 Apr. 2017; accepted 27 June 2017. Date of publication 4 July 2017; date of current version 13 Oct. 2020. (Corresponding author: Wei Zhao.)
Digital Object Identifier no. 10.1109/TSC.2017.2723006

resources for processing and analyzing, and traditional computing paradigms may be incapable of dealing with such scenarios.

In recent years, the emergence of cloud computing has brought many new approaches for resource utilization and management [5]. Cloud manages all resources as a resource pool and provides those resources as online services to end users according to their demand. Such modes can greatly increase the utilization efficiency of resources and, at the same time, save the investment of users on both hardware and software resources [6]. Moreover, big data processing and analysis technologies developed along with cloud computing make data analysis much easier and faster in the IM field [7]. Motivated by these benefits, many researchers are exploring novel cloud based IM technologies to solve the problems above [8]. Until now, most of the work carried out in the interdisciplinary area of IM and cloud computing mainly focuses on the application of cloud computing in IM systems, which can only deal with a few aspects of related problems. Little research has been carried out to build novel IM modes and architectures that can inherit the essence of cloud computing for IM systems.

This paper introduces a novel IMC architecture with detailed system implementations. The architecture abstracts instruments and sensors into resources, and encapsulates frequently used modules and functions into services. Services are deployed in the cloud and users can consume these services on demand. IM applications are also deployed and run in the IMC platform. All IT resources are allocated and managed by the IAAS (Infrastructure as A Service) cloud platform, which will reduce investments for users and also increase resource utilization efficiency. By integrating cloud computing and big data processing technologies, IMC can benefit a lot from advantages such as system scalability, fault tolerance, distributed and parallel computing, and so on. An actual system based on this architecture is implemented using various cloud computing and big data processing frameworks. Applications and experiments are designed to test the system. Results show that the IMC architecture designed in this paper can properly integrate cloud computing with IM technologies and greatly facilitate building, managing and using IM systems.

The remainder of this paper is organized as follows: Section 2 presents related work; Section 3 introduces key concepts of IMC; Section 4 describes the detailed IMC architecture designed by this paper; Section 5 illustrates the implementation of the architecture; Section 6 provides some applications and tests over the IMC system; Section 7 discusses challenges and limitations of IMC; and finally, Section 8 concludes the whole paper.

2 Related Work

Most of the work related to IMC mainly focuses on the following areas: Grid-enabled instrumentation systems (GEIS) [9], sensor clouds [10] and instrumentation clouds [11].

GEIS mainly focuses on converting instruments into grid services [12], so that heterogeneous instrumentation resources can be accessed and managed through a grid, which can facilitate data intensive applications to use various grid resources and provide distributed instrumentation and experiment collaborations over the grid [13]. In GEIS, instruments are abstracted into unified services through middleware technologies and standard models. Typical GEISs include Instrument Element [14] architecture from the GridCC project, common instrument middleware architecture [15], e-infrastructure for remote instrumentation from the DORII project [3] and virtual laboratory architecture [16].

Although GEIS provides a good way for distributed instrumentation and collaborative experiments over the grid, limitations of grid computing, such as complexity, constrained accessibility and so on, prevented it from prevailing among scientific and industrial areas. However, some of the research work in GEIS, regarding common aspects, can guide the study of IMC. Data retrieval and transmission approach in distributed IM systems is an important one of those aspects. Through comprehensive research, [12] and [17] demonstrated that publish/subscribe mechanisms are more efficient for real-time data collecting and transmitting in a distributed IM environment. Thus, in the work of this paper, a message-based publish/subscribe mechanism is adopted as the information and data dissemination method.

Just like IMC, sensor cloud is also a very new concept. In [18] a sensor cloud infrastructure is developed and physical sensors are abstracted into virtual sensor objects. Such virtual sensor objects combined with related sensor definition templates can provide measurement services to end users. Besides, service and accounting models are designed, which makes the sensor cloud infrastructure conform to the philosophy of cloud computing. Another project working on cloud-based sensing systems is S4T (Stack4Things) [19]. This project is the base for the #SmartME [20] project, which mainly focuses on morphing a city into a smart city. The S4T project is trying to build up a cloud platform for managing a large number of sensors and actuators deployed in a distributed environment. One of the main ideas of the S4T project is virtualizing sensors and actuators into abstract resources similar to those resources in cloud computing and providing sensing and actuating services through the S4T platform. Thus, the S4T platform is also viable for building a sensor cloud but it is more focused on building a cloud on the edge of a sensor network. Some other researchers also use the terminology 'sensor cloud', but most of them only concentrate on the application of cloud computing technology in sensor control and management [21], [22], [23], [24]. Sensors are much simpler than instruments, however they can also be treated as instruments, only with less functions.

Current studies of instrumentation clouds only bring forward conceptual models and architectures with few details or implementations provided [11], [25], [26]. Some other research work is mainly about applications of cloud computing in massive data storage and processing [27], [28], [29], which, as explained before, only provide solutions for a few of the problems faced by current IM systems.

Compared with the above research, the work of this paper has the following advantages: (1) the IMC platform developed in this paper is more open and easier to access than GEIS; (2) the IMC platform can manage both sensors and instruments, and provide commonly used IM functions as on-demand, scalable services to end users; (3) the IMC system implemented according to the IMC architecture in this paper is a real cloud platform for the IM field, rather

than just an application of cloud computing technologies in the IM field. All these advantages are achieved by adopting a cloud-based resource management and usage mode, and using state-of-the-art technologies from cloud computing and big data fields. Details will be presented in the following sections.

3 Instrumentation and Measurement Cloud

Currently, no standard definition for IMC is presented. This section brings up a relatively standard definition for IMC and related elements by adopting key ideas of cloud computing in the IM field.

3.1 Definition of IMC

The key ideas of cloud computing are: resource abstraction and virtualization, services delivered on demand, and scalability [30]. Owing to these ideas, cloud computing can provide scalable, on-demand hardware and software resources remotely. As introduced in Section 1, traditional IM systems and modes do not own such advantages. To inherit these merits, it is necessary to integrate cloud computing and related big data technologies into IM field, and establish a novel Instrumentation and Measurement Cloud (IMC). Based on our previous work [31], a definition for IMC is brought forward below:

Definition 1. *Instrumentation and Measurement Cloud(IMC) is a model, based upon cloud computing and big data frameworks, for enabling on-demand, convenient, ubiquitous network access to a shared pool of Instrumentation and Measurement(IM) resources(e.g., instruments, sensors, actuators) and services that can be rapidly provided and released with minimal interaction with resource provider and end user.*

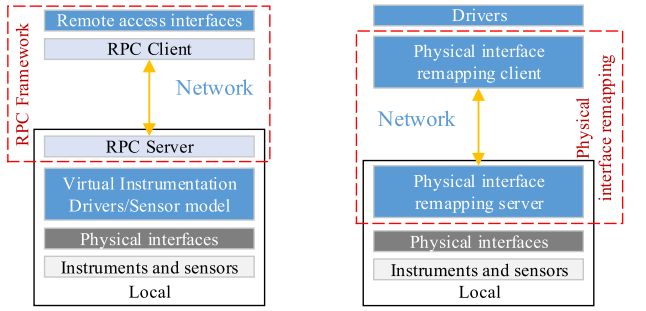
From Definition 1, it can be seen that IMC contains two direct entities, resource and service, and a third hidden entity which is the IM application designed by users. To distinguish these entities from similar concepts in the traditional IM field, IMC resource, IMC service and IMC application are used to feature these three entities of IMC. In this paper, IMC resource, IMC service and IMC application are also called IMC elements in general.

3.2 Definition of IMC Resource

Definition 2. *Instrumentation and Measurement Cloud resource stands for virtualized instrument, sensor or actuator resources that are connected to IMC through the network for online sharing and management.*

As can be seen from the above definition, computing resources are not included in IMC resources. The reason why we define IMC resource this way is that computing resources, such as the CPU, storage and the network, are managed and provided by cloud computing frameworks of IMC, and, to IMC, they play the role of service.

Unlike computing, storage and networking resources, instruments and sensors have heterogeneous software and hardware architectures, which means it is very difficult to virtualize them into a unified resource pool and at the same time keep their full features. However, thanks to virtual instrumentation (VI) technology and standard sensor models, many of the modern instruments and sensors can provide unified access interfaces. However, such interfaces are designed just for local drivers. To virtualize those IM devices into IMC resources, interface remapping through the network is required. Generally, there are two ways to remap the interfaces, as shown in Fig. 1.



(a) Software interface remapping (b) Physical interface remapping

Fig. 1. Remote instrument or sensor interface remapping.

Fig. 1a shows a software interface remapping scheme using the remote procedure call (RPC) approach. Such a scheme is more data-oriented, since it mainly focuses on manipulating data exchanged between physical devices and up-level applications. This method is easier but less flexible. For different VI frameworks or sensor models, corresponding RPC modules should be developed.

The second method, illustrated in Fig. 1b, remotely maps physical interfaces, such as USB [32], RS-232, GPIB and many others, to the cloud side, thus IM device connections to those interfaces will be forwarded to the cloud. Physical interface remapping is a device-oriented design which is more concerned about the device itself rather than the data generated from the device and it supports full features of the device. However, implementation of this method is much more difficult, especially for high-speed interfaces such as PCIE, than that of the software interface remapping approach. Furthermore, in this method, each IMC resource should be attached to a VM in the cloud. An IM system based on physical interface remapping is more like a traditional IM system. Unless there are special requirements, it is better to use software interface remapping for IM device virtualization.

3.3 Definition of IMC Service

Definition 3. *Instrumentation and Measurement Cloud service stands for online, scalable, shared IM functions, cloud computing resource services and big data processing services that can be consumed through IMC by IMC applications on demand.*

The definition of IMC service is close to the concept of SAAS (Software as a Service) in cloud computing [33]. In IMC, commonly used IM functions are encapsulated into online services. Users can consume those services on demand and all those services are running in IMC. When consuming IMC services, users just need to send data to input interfaces of IMC services and retrieve results from output interfaces.

3.4 Definition of IMC Application

Definition 4. *Instrumentation and Measurement Cloud application is the application program that consumes IMC resources and IMC services, and carries out custom logics to fulfill user defined IM tasks.*

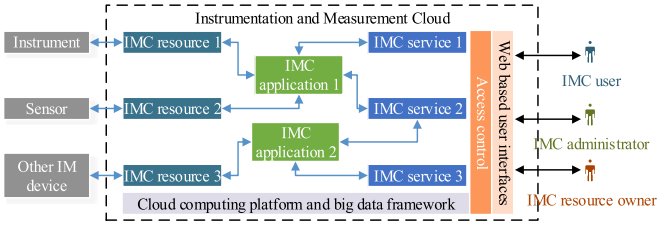


Fig. 2. Instrumentation and measurement mode in the IMC.

Although the IMC platform can provide remote access to IMC services and IMC resources, it still requires the user to organize those services and manipulate related resources to carry out specific IM tasks. Since most of the computation-intensive data processing procedures can be implemented into IMC services deployed in IMC platforms, IMC applications are normally light-weighted. By developing web-based online IDE (Integrated Development Environment), IMC can provide PAAS (Platform as a Service) services to users, so that end users can develop, deploy and run IMC applications online through a web browser. In this case, building a distributed IM system will be much easier, since all IM resources and commonly used IM functions can be obtained online through IMC.

3.5 Instrumentation and Measurement Mode in the IMC

The instrumentation and measurement mode in the IMC can be depicted as in Fig. 2.

In Fig. 2, there is an IMC user, an IMC administrator and an IMC resource owner. The IMC user normally consumes IMC resources and IMC services, and develops IMC applications to carry out IM tasks. The IMC administrator is responsible for building and maintaining the IMC platform. All IMC resources are provided and maintained by IMC resource owners. While IMC services can be developed by any of them, it is the IMC administrator's responsibility to check the quality of services and decide whether to deploy and publish them or not. As shown in Fig. 2, with a web-based user interface and access control, IMC users can request IMC resources and IMC services, and develop IMC applications online. By deploying IMC applications in the IMC platform, IMC users no longer need to invest in, establish and maintain the whole ICT (Information and Communication Technology) system for their IM tasks. Instead, they can use resources and services provided by the IMC platform according to their need.

The above definitions have clarified the basic function requirements and characteristics of IMC. In the following section, a novel IMC architecture, which owns the above important characteristics, will be presented.

4 Novel Architecture for Instrumentation and Measurement Cloud

The overall IMC architecture designed in this paper is shown in Fig. 3. This architecture mainly consists of six parts, which are coordination system, message broker, IMC resource agent, IMC service pool, IMC application executor and IMC manager. Details of each part will be presented in the following sections.

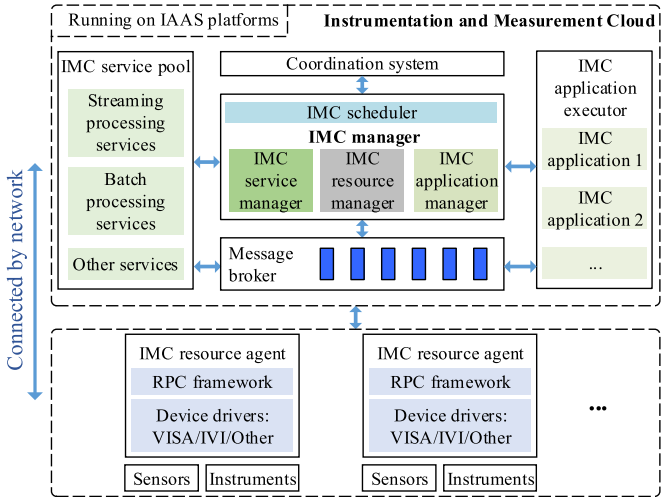


Fig. 3. Novel architecture for IMC.

4.1 Coordination System and Message Broker

The Coordination system records all configuration data and management data of living IMC services, IMC resources and IMC applications. As the IMC architecture in Fig. 3 is distributed, the coordination system should have a distributed synchronization mechanism for data manipulation. The coordination system should also support event notification, so that events from the three IMC elements can be discovered across the architecture and corresponding actions can be taken.

The message broker is the core component for data transmission. Data exchanges between IMC applications, IMC services and IMC resources are mainly achieved by the message broker. As illustrated in Section 2, the effective and efficient way to transmit data in a distributed environment is via a publish/subscribe based messaging mechanism, thus a publish/subscribe based message broker is a good choice for data transmission in IMC. To transmit data over the message broker, a related client of the message broker should be integrated into IMC elements.

4.2 IMC Resource Agent

The IMC resource agent is responsible for instrument and sensor virtualization and IMC resource registration.

As illustrated in Section 3.2, instruments and sensors can be virtualized through VI frameworks and standard sensor models. By virtualization, instruments and sensors are encapsulated into IMC resources with standard access interfaces. Assisted by RPC frameworks, these interfaces can be called remotely from IMC applications in the cloud, which will bring much convenience for building distributed IM systems.

Once virtualized, these IM resources can be registered into IMC by the IMC resource agent, so that users can use them over networks. To use IMC resources, users should first make a reservation for each resource through the IMC manager. After reservation, an access id with start and end time stamps will be allocated to IMC applications and the IMC resource agent, and also, the entry of the resource, such as the URL of the IMC resource agent, will be sent to IMC applications. When IMC applications want to access the reserved resources, first, they will have to send the

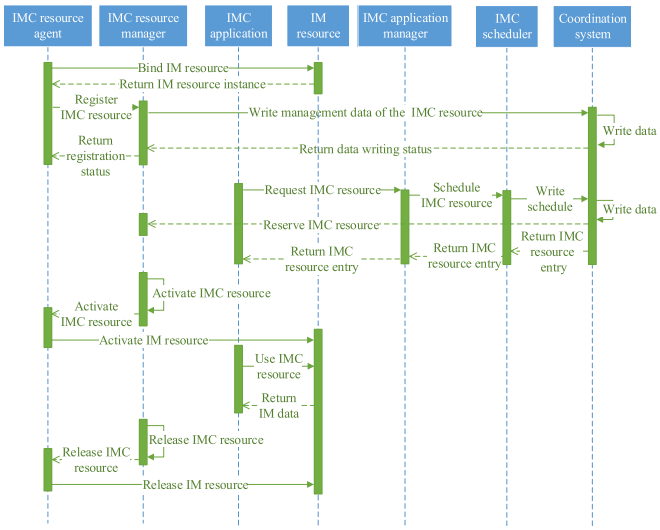


Fig. 4. Steps required for registering and consuming IMC resources in the IMC.

access id to the IMC resource agent through that entry, and the IMC resource agent will then check if it also has the same access id and whether the current time is between the reserved time span. If all requirements are satisfied, the IMC resource agent will allocate a resource handler for the IMC application and allow the IMC application to use the IMC resource through RPC interfaces for instrumentation and measurement tasks.

The complete procedure for registering and using IMC resources in the IMC is depicted through the UML activity diagram shown in Fig. 4.

4.3 IMC Service Pool

In IMC, services represent modularized function blocks implemented in big data analyzing and cloud computing frameworks. Such IMC services include stream data processing modules, batch processing modules and other function modules. IMC services are normally developed and deployed in parallel with distributed big data processing platforms. Each type of service can serve multiple IM applications and the platform or framework running these services will provide scaling, parallel processing and fault tolerance abilities. Services and applications in IMC are connected by a publish/subscribe based message broker.

The message broker is used to transmit data between IMC services, IMC resources and IMC applications. Currently, many message brokers support clustering, which means brokers can support fault tolerance. However, overheads from message headings and message routing can degrade data transmission performance. To deal with this problem, some message brokers support light message headings and a simple message routing scheme. This can increase message processing and transmission speed, but at the same time reduce flexibility and functionality of the broker. Other brokers can provide more flexible control over message transmission and fault tolerance by adding extra information to messages, but this will bring more overheads. IMC service providers should choose the broker according to their needs.

All services in IMC should register themselves through the IMC manager. When registering, the IMC manager will create data entries for each IMC service in the coordination system and write management data of services into those entries. Normally, each IMC service has several input and output interfaces. For input interfaces of a service, the IMC manager will write message broker topics to which they are listening to their data entries and, if IMC applications are to consume this service, they can get those topics through the IMC manager and then publish messages to those topics. To get processed data from the IMC service, IMC applications need to write topics to which their sink modules are listening to data entries of the service's output interfaces.

As for stream data processing services, each of them has several input and output interfaces. Input interfaces will listen to dedicated message topics and, as for output interfaces, they will watch on a cache that stores destination message topics for outputting data.

For batch processing services, file systems and databases constitute the data sources. In most cases, batch processing is an off-line post-processing approach, but IM systems normally deal with real-time stream data; thus, batch-processing services will not be studied in this paper. However, batch-processing services are still indispensable to the whole IMC architecture.

To enable the parallel and distributed computing paradigm and enhance the ability of fault tolerance for IMC services, three key roles should be followed when developing IMC services:

1. Reduce coupling and dependency between data. Only in this way can the service be implemented in a parallel computing paradigm.
2. Make data self-descriptive. This is very important for multiple IMC applications to share the same service instance.
3. Try to avoid state caching for IMC applications in services and use dedicated memory cache systems. Most distributed and parallel cloud computing frameworks support fault tolerance. It means that when some processing nodes go down, the system can still run in a normal state by shifting tasks to other nodes. However, if those nodes cached states of IMC applications that they serve, all these states will be lost and restoring the service process can be difficult, especially for streaming processing applications. Moreover, avoiding state caching in service instances can facilitate online load transfer, which is vital to load balancing.

4.4 IMC Application Executor

The IMC application executor is responsible for running IMC applications that are deployed in the IMC. It often consists of a script interpreter or runtime engine. By deploying IMC applications into IMC, users can save the trouble of maintaining the client side. With proper user interfaces, users can access their IMC applications even through mobile terminals.

4.5 IMC Manager

The kernel of the whole architecture is the IMC manager. The IMC manager contains four main components: the

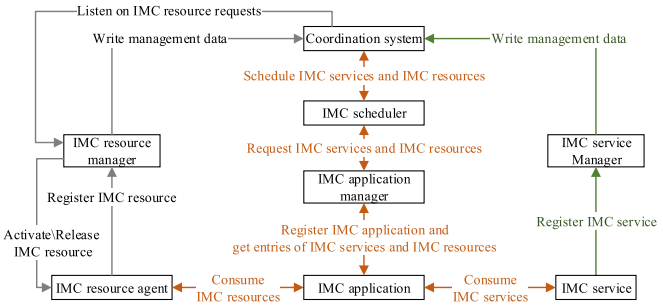


Fig. 5. Details of IMC manager.

resource manager, the service manager, the application manager and the scheduler. All IMC resources, IMC services and IMC applications are registered and managed by the corresponding component of the IMC manager. Fig. 5 shows how the IMC manager works.

As shown in Fig. 5, when registering IMC resources, the resource manager will create a data entry for each IMC resource and store their management data. Under a RPC framework, management data often contains the URL of the resource side RPC server. Another data entry that caches all reservation information of the IMC resource is also created upon registration. Such reservation information will also be sent to the IMC resource agent for authorization purposes. A similar process happens when the service manager is registering services. However, the service manager mainly maintains management data about interfaces of each service. Service and resource requests from IMC applications are processed by the application manager. When IMC applications request services or resources, the application manager will query the coordination system, get all related management data and send these data back to applications. At the same time, some of the message broker's context of sink modules in IMC applications will be written into data entries to which output interfaces of services are listening.

The tasks of the scheduler are IMC resource and service scheduling, load balancing and scaling of IMC services. When an IMC application requests IMC resources, the scheduler will call related algorithms to calculate a valid time span for each IMC resource and make reservations. Load balancing and service scaling is done by online load transfer and scaling the cluster that runs the service. However, online load transfer needs the support from both the framework that runs the IMC service and the IMC service itself.

Whenever an IMC application is registered in the IMC, the IMC application manager will record all consumed services and resources. If the state of services or resources changes, an event will be sent to the application manager to trigger the state transition process of the IMC application. In addition, a change of the IMC application state can trigger state transition of related IMC resources. State transition models for IMC service, IMC resource and IMC application in IMC are shown in Fig. 6.

Fig. 6 shows only very basic state transition models for the IMC architecture; however, implementation of the IMC architecture will need detailed models to handle situations that are more complicated.

To demonstrate the feasibility and advantages of the IMC architecture brought forward by this paper, a practical system implemented upon this architecture is presented.

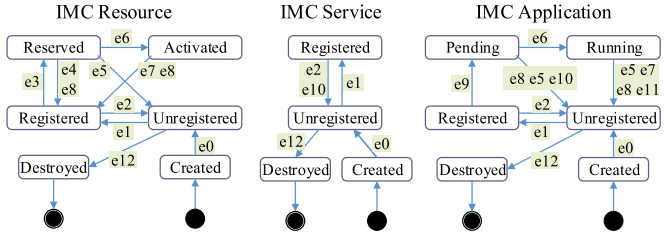


Fig. 6. State transition models of IMC service, IMC resource and IMC application in IMC. In the figure events are: e0-add element, e1-register, e2-unregister, e3-reserve, e4-cancel reservation, e5-resource invalid, e6-resrouce available, e7-resource expired, e8-application invalid, e9-application start, e10-service invalid, e11-application terminated, e12-destroy element.

5 Implementation of the IMC Architecture

5.1 Coordination System and Message Broker

The coordination system is used to record important configuration data and management data about IMC resources, IMC services and IMC applications in the IMC. Here, Zookeeper, which is a distributed coordination system with fault tolerance ability, is utilized to store various data. Zookeeper uses a data structure called Zookeeper path that is similar to the directory of a file system and where each node of a path can store data.

Various message brokers can be used in the IMC and, in this paper, Rabbitmq is adopted just for demonstration.

5.2 VISA Based IMC Resource Agent

First, to verify that the architecture is viable for instrument and sensor virtualization, a Virtual Instrument Software Architecture (VISA) based IMC resource agent is implemented. The agent uses Apache Thrift RPC framework and all VISA driver interfaces are remotely mapped. Since Thrift supports cross language service development, it is also used as the service framework between all servers and clients in the IMC. Fig. 7 shows the details of the implemented IMC resource agent.

As shown in Fig. 7, instrument resources are registered through IMC resource management RPC services, which are implemented in the Thrift framework. To access the instrument, each IMC resource agent needs to run a VISA RPC server and it wraps all VISA driver interfaces. However, those interfaces are extended to include a third parameter, which is the access ID. Such access ID contains both the name of the IMC resource and the reserved time span. The IMC resource agent will also store a set of <access ID, VISA instrument resource> maps and these maps are built

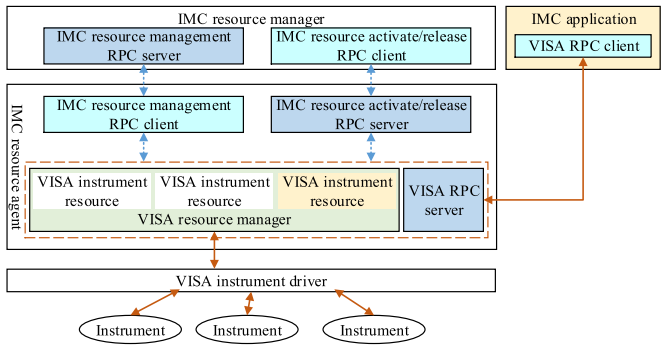


Fig. 7. Implementation of the IMC resource agent.

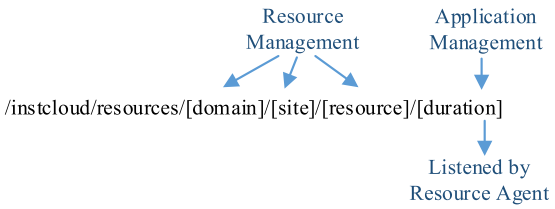


Fig. 8. Management data for the IMC resource.

up when IMC applications request IMC resources managed by this IMC resource agent.

Once the IMC application in the IMC needs to control an IMC resource, it will make a remote call with the access ID. On the IMC resource agent side, the agent will get the corresponding VISA instrument resource through this ID and call a local VISA instrument driver to control the instrument and then return the result to the IMC application. To make the agent as independent with OS platforms as possible, pyvisa, which is a python implemented frontend of VISA driver, is used as the VISA resource manager. Although instruments and sensors are diverse in their hardware architectures and software interfaces, the similar implementation method can be applied to virtualize them into an IMC resource as long as their interfaces are available.

As for IMC resources, the Zookeeper data paths used to record their information are shown in Fig. 8. The */instcloud/resources* path is the root path for IMC resource management. When registering an IMC resource, the resource manager of the IMC manager will create a data path according to the domain, site and the resource name. Here domain and site are used to constitute a two-level naming convention so that resource management can be more convenient and flexible. Whenever a reservation is made for an IMC resource, a duration node with a start and an end timestamp will be added as a child to the resource node as long as the reservation is valid. The IMC resource manager will listen to such child-adding event and call the PRC client to activate the IMC resource.

5.3 Storm Based IMC Service for Stream Data Processing

The next stage is to implement IM services. As explained before, IM tasks normally need to process real-time stream data, thus a stream data processing engine is required and related IM function modules need to be developed. In the work of this paper, Apache Storm [34] is used as the real-time stream data processing engine. Storm is a distribute parallel stream data processing engine, whose maximum processing speed can reach around 1 million messages per second, with fault tolerance ability. Storm contains two types of components, which are Spout and Bolt. Spout is the data source and it can be connected to a message broker to subscribe message topics. Bolt is the process component. Spouts and Bolts compose a Topology, which carries out the stream data processing logic. Each Spout or Bolt can run multiple instances so that they can process data in parallel. In the IMC, topologies are wrapped into IMC services, with certain Spouts functioning as input interfaces and some Bolts as output interfaces. Fig. 9 presents a simple example for computing electrical power to show how services in the

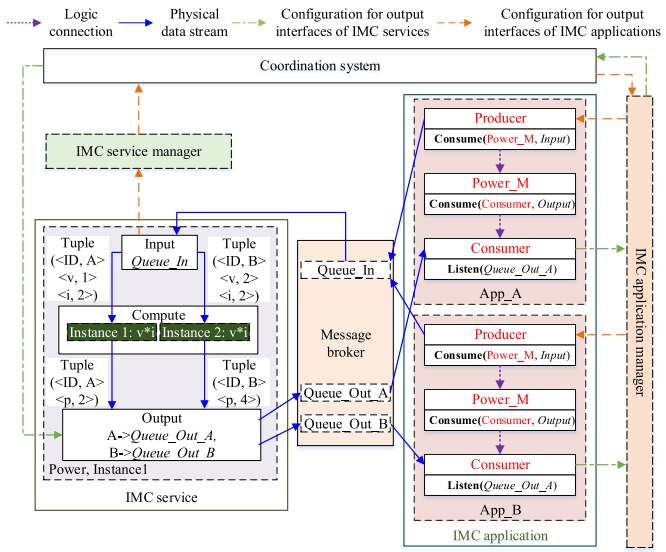


Fig. 9. An IMC service implemented through Storm to compute electrical power

IMC interact with IMC applications and the IMC manager in the cloud.

In Fig. 9, input interfaces of an IMC service are implemented by instances of IMCSpout class, which is an extended Spout class with a service management RPC client. When an IMC service is submitted, each IMCSpout instance will register the context of the message topic that this IMCSpout is listening to through the RPC client. Since Rabbitmq is used as the message broker, the detailed context of a message topic is wrapped into the RMQContext class. To consume an IMC service, an IMC application just needs to get the RMQContext instance of each input interface, create a Rabbitmq client corresponding to that context, and send data to the IMC service input interface through that Rabbitmq client.

Output interfaces are instances of an extended Bolt class named IMCBolt class with a service management client, and they will also create management data entries in Zookeeper and listen to those data paths. However, it is the IMC application manager that is responsible for writing message topic contexts to those paths. Whenever data on those paths are updated, output interfaces of an IMC service will update destination contexts. Each destination context in an output interface is combined with a session ID and a module ID, which will be introduced in the following part, and each message passed to an output interface will also contain these IDs. With these IDs, output interfaces will know which message topic the data should be sent to. Data paths for IMC services are shown in Fig. 10.

In Fig. 10, */instcloud/services* is the root data path for IMC service management. Under the root path are service name and instance name data path. Children under */in* node are input interfaces of the IMC service and, similarly, children under */out* node are output interfaces. Each child node under */in* node will store a Transport Context which records the context of the message topic that the interface is listening to. Each child node under */out* will store a Transport URL that tells the output interface which message broker is used for data transmission. Under each node of output interface, there are child nodes representing IMC

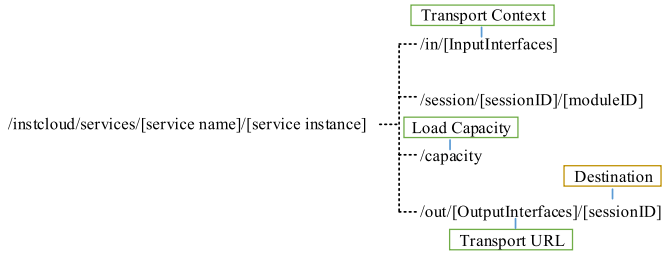


Fig. 10. Management data for IMC services..

application sessions that consume output data from the output interface and each of the child nodes will store a Destination Object that records contexts of message topics for output data transmission.

The */session* node and its children under each topology or service instance node are used for IMC service state management. For example, when an IMC application session releases a service, the corresponding session ID and module ID will be deleted and such deleting events will be listened to by the IMC service manager and related IMC service instances. Once such event happens, both the IMC service manager and related IMC service instances will clear cached data for that session.

The */capacity* node stores load capacity of this service and the IMC manager will use this data for load balance and scaling.

5.4 IMC Application Developing Mode

IMC applications are implemented through text programing language and currently only Java APIs (Application Programming Interfaces) have been developed. Four classes are defined according to entities in the IMC, which are ICResource, ICService, StreamProducer and StreamConsumer. ICResource and ICService are wrappers of the IMC resource and the IMC service respectively.

When creating an ICResource object, the domain, site, resource name and time span should be specified, but for the ICService object, only service name is required. The ICResource class is normally wrapped with a RPC client that is used to control IMC resources.

StreamProducer is used to publish data to the input interfaces of an IMC service, while StreamConsumer is responsible for receiving data from an IMC service. However, all related message broker contexts are automatically set when submitting the IMC application. A complete IM process is wrapped into a session task and all related IMC resources, IMC services and other modules are managed through a Session object. All the four types of components in an IMC application session have their module IDs and each session has a universally unique ID. However, each module ID is only unique to a session. Fig. 11 shows a diagram of a simple IMC application, which is also referred to in Fig. 9, just for illustration.

In Fig. 11, there is a consumeService method that is used to decide which module and interface the data of the current module should be sent to. For the StreamProducer object, the name of the next module and the interface should also be provided, while for the consumeService method of the ICService object, the name of the service output interface and the next module and its input interface, if any, should be defined.

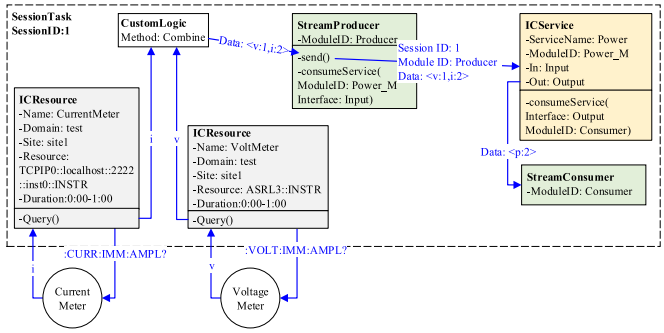


Fig. 11. An IMC application example.

5.5 Implementation of the IMC Manager

Implementation of the IMC manager is shown in Fig. 12. In Fig. 12, the IMC manager needs to interact with Zookeeper, IMC resource agents, IMC applications and IMC services. In the implementation of this paper, Curator Framework is used to interact with Zookeeper. Most of the other remote management operations are carried out through the Apache Thrift RPC framework.

To obtain load capacities of Storm-based IMC services and schedule IT resources for those IMC services, the IMC scheduler will call the Storm REST (Representational State Transfer) API. Each module of the IMC manager in Fig. 12 can run a thread pool for its RPC server, so that it can serve multiple RPC clients.

Currently, the IMC manager is implemented in a centralized way just to verify the feasibility and functionality of the IMC architecture in this paper. As shown in Fig. 12, the IMC resource manager, the IMC service manager, the IMC application manager and the IMC scheduler are loosely coupled through an event bus. In this way, the IMC manager can manage the state of each IMC element according to Fig. 6.

However, each module of the IMC manager is relatively independent, thus it will be very easy to implement the IMC manager in a distributed way, e.g., substitute event bus with message broker.

6 Application and Experiment

To test the designed IMC architecture, an application for power system state estimation (SE) [35] is developed based on the implemented system.

6.1 Distributed Parallel Power System State Estimation

In a power system, not all states of the system are monitored and some measured states may not be correct. However, all real states of a power system should always obey the law of electric circuits. Thus, using these electric circuit laws and

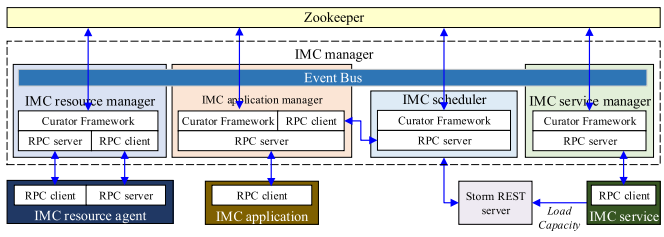


Fig. 12. Implementation of the IMC manager.

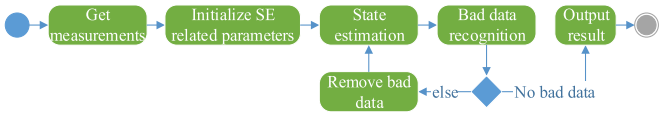


Fig. 13. Power system state estimation procedure.

measured data, a group of measurement equations can be built up. With these equations, a mathematic model can be developed to estimate the true states of the whole power system. The most fundamental state estimation method is the WLS (Weighted least square) method [36], also referred to as the NE (Normal equation) method. The mathematical model for the WLS method can be formulated through Eqs. (1) - (4) below:

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{v} \tag{1}$$

where z is the m-dimensional measurement vector, $\mathbf{x}$ is the $2n-2$ dimensional state variable vector, $\mathbf{v}$ is the m-dimensional measurement error vector and $\mathbf{h}(\mathbf{x})$ is the relation function of measurements and the state variable $\mathbf{x}$.

$$\begin{cases} J(\mathbf{x}) = [\mathbf{z} - \mathbf{h}(\mathbf{x})]^{\mathrm{T}} \mathbf{W} [\mathbf{z} - \mathbf{h}(\mathbf{x})] \\ \mathbf{W} = \mathrm{diag}\left[\frac{1}{\sigma_1^2}, \ldots, \frac{1}{\sigma_i^2}, \ldots, \frac{1}{\sigma_m^2}\right], \end{cases} \tag{2}$$

where $J(\mathbf{x})$ is the objective function, $\mathbf{W}$ is the $m \times m$ diagonal weight matrix, and σ_i^2 is the covariance of the ith measurement error,

$$\Delta \mathbf{z}(\mathbf{x}) = \mathbf{z} - \mathbf{h}(\mathbf{x}) \tag{3}$$

where $\Delta \mathbf{z}(\mathbf{x})$ is the measurement residual vector,

$$\begin{cases} \mathbf{H}^{\mathrm{T}}(\mathbf{x}_k)\mathbf{W}\mathbf{H}(\mathbf{x}_k)\Delta \mathbf{x}_{k+1} = \mathbf{H}^{\mathrm{T}}(\mathbf{x}_k)\mathbf{W}\Delta \mathbf{z}(\mathbf{x}_k) \\ \mathbf{H}(\mathbf{x}_k) = \frac{\partial \mathrm{h}(\mathbf{x}_k)}{\partial \mathrm{x}}, \end{cases} \tag{4}$$

where $\mathbf{H}(\mathbf{x}_k)$ is the value of the Jacobian matrix of $\mathbf{h}(\mathbf{x})$ in which $\mathbf{x} = \mathbf{x}_k$, $\Delta \mathbf{x}_{k+1}$ is the correction vector for the estimated state $\mathbf{x}_k$ and k is the iteration index.

The WLS state estimation method is carried out by iterating (4).

As there are bad data among measurements, a bad data recognition process is required to find and eliminate bad measurements. A commonly used method is the normalized residual recognition method [37]. This method uses Eqs. (5) and (6) below to detect and identify bad measurement data,

$$\mathbf{r}_{\mathrm{N}} = \left(\mathbf{W} - \mathbf{H}(\mathbf{x})\left(\mathbf{H}^{\mathrm{T}}(\mathbf{x})\mathbf{W}^{-1}\mathbf{H}(\mathbf{x})\right)^{-1}\mathbf{H}^{\mathrm{T}}(\mathbf{x})\right)^{-\frac{1}{2}} \Delta \mathbf{z}(\mathbf{x}) \tag{5}$$

where $\mathbf{r}_{\mathrm{N}}$ is the normalized residual vector for measurement vector z,

$$\gamma = \left(\sum_{i=1}^{m} \sigma_i^2\right)^{\frac{1}{2}}, \tag{6}$$

where γ is the threshold for bad data recognition.

If the ith element $r_{\mathrm{N}i}$ of $\mathbf{r}_{\mathrm{N}}$ is greater than γ, then the corresponding measurement z_i is taken as the bad data suspect. The measurement, which has the largest normalized residual larger than γ, is taken as the prime bad data suspect. Once bad measurement data are detected, they should be discarded and the state estimation process should restart.

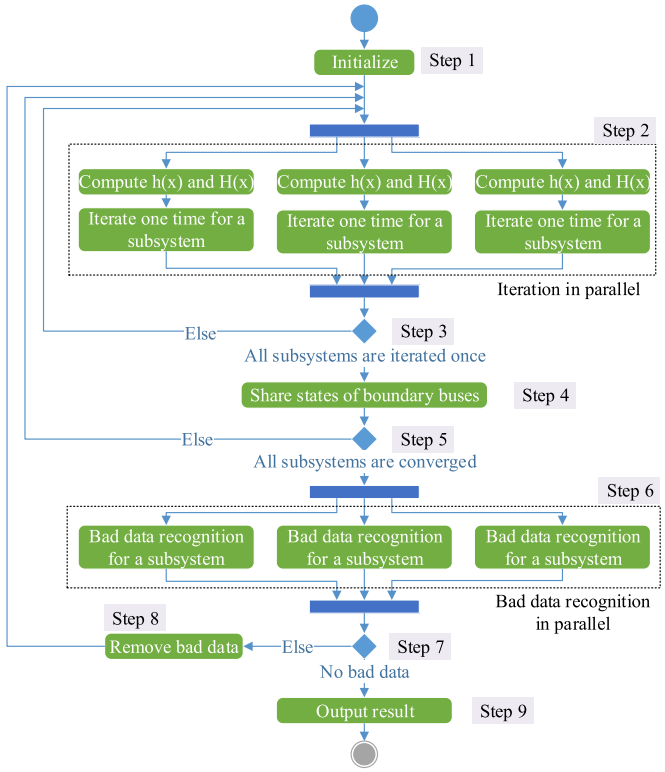


Fig. 14. Parallel state estimation for a large power system.

Normally, bad measurement data will not be discarded at one time, instead, only the bad data that has the prime suspect is discarded. The procedure of state estimation with bad data recognition for power systems is shown in Fig. 13.

In Fig. 13, the state estimation step is actually carrying out the iteration process formulated in (4), while the bad data recognition step is calculating $\mathbf{r}_{\mathrm{N}}$ and comparing its elements to γ.

Nevertheless, the estimation process is quite computation-intensive and consumes lots of memory, especially when the system is large. Still, the state estimation process is required to be as fast as possible. To achieve this goal, the parallel state estimation method [38] is introduced.

The parallel state estimation method is based on system partitioning. To implement a parallel state estimation algorithm, a large system should, first, be partitioned into several subsystems where all subsystems can estimate their own state in parallel. Each subsystem can use the aforementioned WLS method to estimate state except that the iteration process is different. In a parallel state estimation algorithm, after each iteration, all subsystems should share the states of boundary buses before the next iteration. The basic procedure of parallel state estimation algorithm is depicted in Fig. 14.

In Fig. 14, Step 2 carries out the iteration process formulated in (4) while Step 6 calculates $\mathbf{r}_{\mathrm{N}}$ and compares its elements to γ. Iteration and bad data recognition tasks from multiple subsystems can be distributed in a cluster to accelerate the overall estimation speed.

6.2 IMC-Based Power System State Estimation

Since the IMC platform can greatly facilitate building distributed IM systems and provide efficient big data processing frameworks, this section tries to establish an IMC-based power system state estimation system. Such a system will

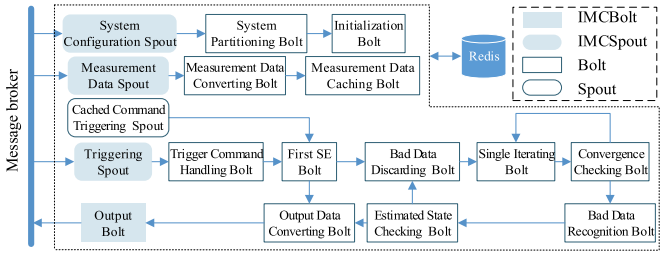


Fig. 15. Storm-based IMC service for power system state estimation.

not only make state estimation easier and more efficient, but can also test the functionalities of the IMC architecture brought up in this paper.

According to the IMC architecture elaborated on before, three aspects should be considered when developing an IMC-based state estimation system: (1) Virtualizing the measurement system into an IMC resource; (2) Implementing the parallel state estimation algorithm into an IMC service; (3) Developing the state estimation IMC application.

(1) Virtualizing the Measurement System Into an IMC Resource. There are eight types of electric variables that need to be measured for state estimation. Since the test cannot be carried out in a real power system for safety reasons, all measurements are simulated by adding Gaussian noise to system power flow data.

As the implemented IMC resource agent is based on VISA, here, a meter that conforms to VISA standard is implemented for each power system through pyvisa-sim, which is an instrument simulation software for pyvisa, and custom defined instrument commands, such as ?MEA, are used to retrieve measurement data from the simulated meter. In this way, the measurement system of each power system can be virtualized into an IMC resource. However, in practical applications, a corresponding IMC resource agent should be developed according to the physical measurement system.

Once the measurement system of a power grid is virtualized into an IMC resource, a state estimation IMC application can retrieve all measured data by sending instrumentation commands through the RPC framework.

(2) Implementing the Parallel State Estimation Algorithm Into an IMC Service. The parallel state estimation algorithm is implemented into an IMC service, which can be consumed by different state estimation IMC applications to estimate states of multiple power systems simultaneously. According to Fig. 14, a Storm-based state estimation IMC service can be developed as shown in Fig. 15.

The IMC service mainly contains three subsystems. The first one is the system partitioning subsystem. This subsystem receives the configuration data of a power system from the SE IMC application through the System Configuring Spout. It is responsible for partitioning a large power system into small subsystems and initializing various parameters for SE. In this test, KaHIP [39], which is an effective and fast graph partitioning tool, is used to partition the power system. Also the topology is implemented through JAVA; however, JAVA is not quite suitable for matrix computation, thus all matrix-related computations are carried out by Matlab. Partitioned data and initialized parameters are all stored in Redis [40], which is an in-memory database, so that computation processes can be separated from data, which is one of the rules for developing an IMC service. Redis uses the key-value data structure, and it is fast and very suitable for big data processing. As an IMC service is shared by multiple IMC applications, when caching data, a key should be attached to each data to identify its origin and that is why Redis is chosen as the caching database. Here, most of the intermediate data are cached in Redis.

The second subsystem is the measurement subsystem. It converts raw measurement data into formatted data, so that they can be used for state estimation. Raw measurement data are acquired from the IMC resource corresponding to the virtualized measurement system of a power system by the SE IMC application, and then sent to the Measurement Data Spout through the message broker. The Measurement Data Converting Bolt will convert the data and then send the converted data to the Measurement Data Caching Bolt to store the data into Redis.

The third subsystem is the SE subsystem. This subsystem implements the parallel SE algorithm as shown in Fig. 14. Whenever a power system needs to estimate state, it can just send a trigger command with the ID of the power system to the SE subsystem through the Triggering Spout. After receiving the trigger command, the Trigger Command Handling Bolt of the SE subsystem will check if the power system that requested SE is currently being estimated. If so, the Trigger Command Handling Bolt will cache the command for the latter SE triggering, otherwise it forwards the command to the First SE Bolt. The Cached Command Triggering Spout will check all cached commands similar to the Command Handling Bolt periodically and send a valid command to the First SE Bolt. The First SE Bolt will then compute $J(\mathbf{x})$ and compare it to the corresponding threshold to see if the initial state is the estimated state. If $J(\mathbf{x})$ is below the threshold, the initial state is the estimated state and it will be forwarded to the Output Data Converting Bolt to convert the data for output, otherwise the SE trigger command will be sent to the Bad Data Discarding Bolt to start a new state estimation process.

The Single Iterating Bolt, the Convergence Checking Bolt, the Bad Data Recognition Bolt, the Estimated State Checking Bolt and the Bad Data Discarding Bolt implement step 2, steps 3-5, step 6, step 7 and step 8 in Fig. 14, respectively. However, there are three main differences.

First, the Bad Data Discarding Bolt has two inputs, which are from the First SE Bolt and the Estimated State Checking Bolt. If the input is from the First SE Bolt, the Bad Data Discarding Bolt will do nothing but send the input command to the Single Iterating Bolt, otherwise it will discard bad measurement data and send a command to the Single Iterating Bolt for new SE.

Second, as estimated states from each iteration are cached in Redis, all states will be updated immediately, which means states of boundary buses are shared in time by the Single Iterating Bolt and that is why no Bolt for step 4 in Fig. 14 is implemented.

Third, each of the Bolts in the SE subsystem can serve different power systems and do not bind to one specific power system. Such a loosely coupled processing characteristic is achieved by caching data in Redis and retrieving

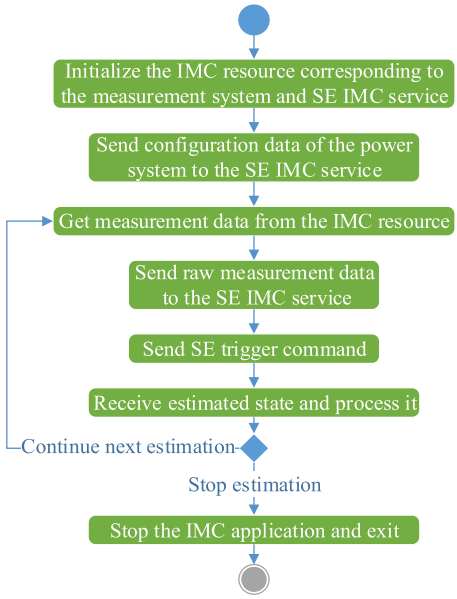


Fig. 16. Activity diagram of the SE IMC application.

corresponding data of each power system through keys such as the ID of a power system.

Parallel iteration and bad data estimation are implemented by running multiple instances of corresponding Bolts. Once the state of a power system is estimated, it will be sent back to the corresponding SE IMC application through the Output Bolt.

Once the topology of the SE IMC service is developed, it can be deployed onto the IMC platform, and the SE IMC service can be started.

(3) Developing the State Estimation IMC Application. With the measurement system being virtualized and the SE IMC service running, to carry out state estimation, end users just need to develop a very simple SE IMC application to acquire measurement data of a power system from the corresponding IMC resource and send those data to the SE IMC service for SE online. Detailed activities of the SE IMC application are shown in Fig. 16.

In Fig. 16, IMC resources and IMC services are defined through instances of ICResource class and ICService class respectively. Data are sent by instances of StreamProducer class and are received by instances of StreamConsumer class.

6.3 System Deployment and Test Scenarios

The whole system runs on an Openstack IaaS cloud platform. Each VM that runs a supervisor node of Storm is configured with E3-12XX/2.3 GHz CPU and 8 GB RAM.

VMs for other nodes in the IMC platform are configured with the same type of CPU as Storm supervisor node but with only 4GB RAM. Detailed physical and logical configurations of the system are shown in Fig. 17.

In this test, case data from Matpower [41] are used and two test scenarios are set. In the first test scenario, state estimation for case3120sp is tested and the SE IMC service is exclusive to case3120sp. The number in the case name represents the number of buses in the system. This scenario is used to test the functionality of the implemented IMC platform to test the IMC architecture brought up in this paper. In the second test scenario, state estimations for

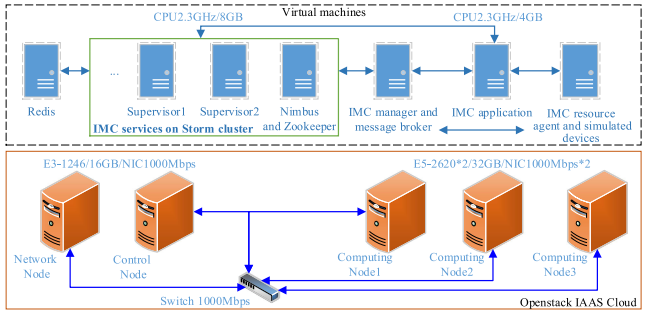


Fig. 17. Detailed configurations of the IMC system.

case2869pegase, case3012wp and case3120sp are tested simultaneously and the SE IMC service is shared by multiple cases. This scenario is used to test the service sharing ability of the IMC architecture. Systems of these cases are very large in the electrical engineering field and, when running the parallel SE algorithm, each system is split into several subsystems. The number of buses in each subsystem is limited to 300 when splitting the system.

The most computation intensive Bolts are the Single Iterating Bolt and the Bad Data Recognition Bolt, and, while these two Bolts are under full load, all load capacities of other Bolts in Fig. 15 are less than 5 percent. Thus, multiple workers are set for the instances of these two Bolts. To find out which Bolt plays a more important role in determining the total estimation time T_{se}, different numbers of workers are set for each of the Bolts.

6.4 Test Results

The result for the first test scenario is shown in Fig. 18.

In Fig. 18, *PB* is the number of workers for the Bad Data Recognition Bolt and *PE* is the number of workers for the Single Iterating Bolt. Fig. 18 shows that, when increasing *PB*, estimation time will reduce dramatically. Fig. 18 also shows that, when *PB* is fixed, changing *PE* does not effectively influence the estimation time.

However, when PB increases to 12 or larger, the estimation time does not decrease any more. This phenomenon is caused by the overheads from distributed computing, such as data transmission delay between computing nodes.

Fig. 18 demonstrates that the bad measurement data recognition process is much more time-consuming than the estimation process. Fig. 18 also shows that sometimes increasing *PB* or *PE* may cause performance degradation and that is because instances of Bolt may distribute across different computing nodes. When more directly connected Bolt instances are running on the same node, communication delay can be reduced. However, if more directly connected Bolt instances distribute across different nodes, overheads from communication will degrade overall

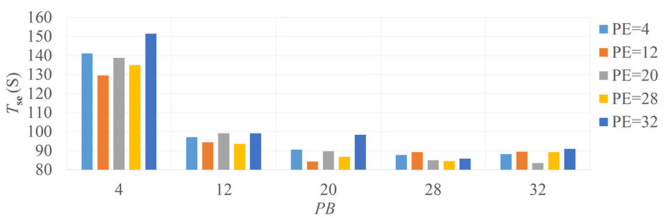


Fig. 18. State estimation time for case3120sp using the IMC platform.

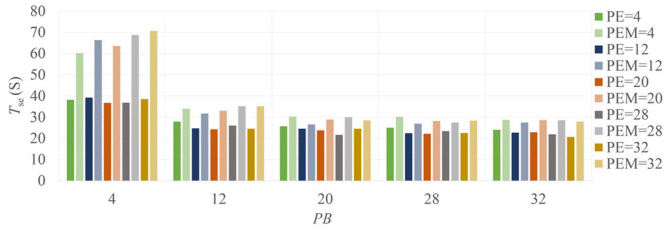


Fig. 19. Comparing T_{se} of case3120sp in exclusive service mode and in shared service mode.

performance. Currently, the distribution of Bolt instances is carried out automatically by Storm and that is the reason for performance fluctuation in Fig. 18. A similar phenomenon will also happen in the second test scenario.

The result of the first test shows that the IMC architecture brought up in this paper is feasible and satisfies the basic functional requirements defined in Section 3.

In the second test scenario, the results for each case alone are similar to Fig. 18; thus, those results are not presented. However, comparing the estimation time of case3120sp in the two different test scenarios is important, since it can reveal some characteristics of shared IMC services. Fig. 19 shows T_{se} of case3120sp in exclusive service mode and in shared service mode.

In Fig. 19, *PEM* designates the number of workers for the Single Iterating Bolt in the shared service mode. Fig. 19 shows that when *PB* is small, which means computing resource for the SE IMC service is inadequate, T_{se} is much higher in shared service mode than in exclusive service mode. However, when *PB* increases, T_{se} in shared service mode is only 10-20 percent higher than in exclusive service mode. This demonstrates that, when computing resource is not strained, performance of the SE IMC service in shared service mode is very close to that in exclusive service mode, which means resource utilization efficiency can be greatly increased through service sharing in the IMC platform.

Such a resource utilization efficiency improvement is normally attributed to the overheads from synchronization of the parallel algorithm. For example, in the parallel SE algorithm shown in Fig. 14, step 2 or step 6 has to wait until all subsystems are processed to continue, and during this period, some of the computing resource will be idle if the service implemented upon this parallel algorithm is exclusive. However, if the service is implemented in shared mode, the idle resource can be used for SE of other power systems, thus resource utilization efficiency can be improved.

Comparing the results from the above two test scenarios demonstrates that the IMC architecture in this paper is viable for service sharing that can improve resource utilization efficiency.

7 Discussion

Although the IMC brought up in this paper can greatly facilitate utilization and management of IM resources, limitations and challenges still exist.

First, the IMC is not suitable for those application scenarios that are highly time-critical and require extremely high reliability. Such limitation is caused by the latency and fluctuation of networks, and overheads from message brokers, RPCs and distributed parallel computing frameworks. Second, high-speed and high-precision IM devices normally produce large amounts of data in a very short time, and directly transferring those large amounts of raw data in real time from IM devices to the IMC may be impossible due to the bandwidth limitation of the network. Third, frequent remote procedure calls can bring a lot of overheads, especially for remote IM device control with short intervals.

Currently, a promising solution for the first and second challenges above is adopting fog computing [42] paradigms as a complementary framework between the IMC layer and the physical IM device layer. Fog computing is more focused on proximity to client objectives and end users, which leads to less latency and higher reliability. By preprocessing data locally or through fog computing, the amount of data that need to be transferred over the network can be greatly reduced, which will lower the requirement for the bandwidth of the network. Also, time-critical tasks can be carried out in a fog computing framework and other computation intensive tasks can be shifted to the IMC. In this way, the overall performance and QoS (Quality of Service) can be greatly improved.

To solve the third problem, frequent remote procedure calls can be substituted by direct interaction between local devices and the IMC services. RPCs can just be used to manipulate the interaction process rather than relay data between IM devices and the IMC.

8 Conclusion and Future Work

The instrumentation and measurement cloud can greatly facilitate management of instruments and sensors and, at the same time, allows users to utilize those resources and related IM services on demand remotely.

The IMC architecture brought forward in this paper provides efficient guidance for developing a practical IMC platform. With IM device virtualization and service wrapping, building a remote IM system just requires only very simple coding work. Most of the investment in IT facilities and system development work can be saved. Furthermore, with the ability to scale and to dynamically transfer load, the IMC can increase the utilization efficiency of various resources to a much higher level. Distributed parallel computing paradigms of the IMC will accelerate the processing speed, which brings lots of benefits for large-scale remote IM systems and, also, for analysis of big data coming from huge numbers of instruments and sensors. The application developed upon the implemented system has demonstrated the advantages of the work in this paper.

However, more research work is still required to deal with some challenges. Such challenges include latency and stability of networks, geographic characteristics of the physical object that is being measured, and so on. These challenges are not proprietary to the IMC but common in the remote instrumentation and measurement field. But with more effort, problems caused by these challenges can eventually be solved.

Acknowledgments

This work was funded by the State Key Laboratory of Power System award SKLD15M02, Department of Electrical Engineering, Tsinghua University, Beijing, China.

REFERENCES

[1] M. Bertocco, "Architectures For remote measurement," in *Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements*, F. Davoli, S. Palazzo, and S. Zappatoreed, Eds. New York, NY, USA: Springer, 2006, pp. 349–362.

[2] A. Roversi, A. Conti, D. Dardari, and O. Andrisano, "A WEB-based architecture enabling cooperative telemeasurements," in *Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements*, F. Davoli, S. Palazzo, and S. Zappatoreed, Eds. New York, NY, USA: Springer, 2006, pp. 395–407.

[3] A. Cheptsov, et al., "E-Infrastructure for remote instrumentation," *Comput. Standards Interfaces*, vol. 34, no. 2012, pp. 476–484, 2012.

[4] S. Qin, X. Liu, and L. Bo, "Study on the networked virtual instrument and its application," in *Proc. Intell. Data Acquisition Advanced Comput. Syst. Technol. Appl.*, 2005, pp. 337–339.

[5] Y. Jadeja and K. Modi, "Cloud computing-concepts, architecture and challenges," in *Proc. Int. Conf. Comput. Electron. Electr. Technol.*, 2012, pp. 877–880.

[6] I. Foster, Z. Yong, I. Raicu, and L. Shiyong, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid Comput. Environ. Workshop*, 2008, pp. 1–10.

[7] M. Fazio, M. Paone, A. Puliafito, and M. Villari, "Huge amount of heterogeneous sensed data needs the cloud," in *Proc. 9th Int. Multi-Conf. Syst. Signals Devices*, 2012, pp. 1–6.

[8] W. Rekik, M. Mhiri, and M. Khemakhem, "A smart cloud repository for online instrument," in *Proc. Int. Conf. Educ. e-Learning Innovations*, 2012, pp. 1–4.

[9] A. S. McGough and D. J. Colling, "The GRIDCC Project," in *Proc. 1st Int. Conf. Commun. Syst. Softw. Middleware*, 2006, pp. 1–4.

[10] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," in *Proc. 13th Int. Conf. Netw.-Based Inf. Syst.*, 2010, pp. 1–8.

[11] R. Di Lauro, F. Lucarelli, and R. Montella, "SIaaS-sensing instrument as a service using cloud computing to turn physical instrument into ubiquitous service," in *Proc. IEEE 10th Int. Symp. Parallel Distrib. Process. Appl.*, 2012, pp. 861–862.

[12] F. Lelli, "Bringing instruments to a service-oriented interactive grid," *Ph.D Thesis*, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Venice, Italy, 2007.

[13] G. C. Fox, et al., "Web 2.0 for grids and e-science," in *Grid Enabled Remote Instrumentation*, F. Davoli, N. Meyer, R. Pugliese, and S. Zappatoreed, Eds. New York, NY, USA: Springer, 2009, pp. 409–431.

[14] E. Frizziero, et al., "Instrument element: A new grid component that enables the control of remote instrumentation," in *Proc. 6th IEEE Int. Symp. Cluster Comput. Grid*, 2006, pp. 44–52.

[15] I. M. Atkinson, et al., "Developing CIMA-based cyberinfrastructure for remote access to scientific instruments and collaborative e-Research," in *Proc. Conf. Res. Practice Inf. Technol. Series*, 2007, pp. 3–10.

[16] M. Okon, et al., "Virtual laboratory as a remote and interactive access to the scientific instrumentation embedded in Grid environment," in *Proc. 2nd IEEE Int. Conf. e-Sci. Grid Comput.*, 2006, pp. 1–5.

[17] L. Berruti, F. Davoli, and S. Zappatore, "Performance evaluation of measurement data acquisition mechanisms in a distributed computing environment integrating remote laboratory instrumentation," *Future Generation Comput. Syst.*, vol. 29, no. 2, pp. 460–471, 2013.

[18] M. Yuriyama, T. Kushida, and M. Itakura, "A new model of accelerating service innovation with sensor-cloud infrastructure," in *Proc. Annu. SRII Global Conf.*, 2011, pp. 308–314.

[19] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito, "Stack4Things: Integrating IoT with OpenStack in a smart city context," in *Proc. Int. Conf. Smart Comput. Workshops*, 2014, pp. 21–28.

[20] G. Merlino, D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and A. Al-Anbuky, "A smart city lighting case study on an openstack-powered infrastructure," *Sensors*, vol. 15, no. 7, pp. 16314–16335, 2015.

[21] K. Ahmed and M. Gregory, "Integrating wireless sensor networks with cloud computing," in *Proc. 7th Int. Conf. Mobile Ad-Hoc Sensor Netw.*, 2011, pp. 364–366.

[22] M. S. Aslam, S. Rea, and D. Pesch, "Service provisioning for the WSN cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 962–969.

[23] Y. Byunggu, A. Cuzzocrea, J. Dong, and S. Maydebura, "On managing very large sensor-network data using bigtable," in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 918–922.

[24] Y. Chang Ho, H. Hyuck, J. Hae-Sun, Y. Heon Young, and L. Yong-Woo, "Intelligent management of remote facilities through a ubiquitous cloud middleware," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2009, pp. 65–71.

[25] S. Distefano, G. Merlino, and A. Puliafito, "Sensing and actuation as a service: A new development for clouds," in *Proc. 11th IEEE Int. Symp. Netw. Comput. Appl.*, 2012, pp. 272–275.

[26] Y. Kang, Z. Wei, and H. Song-ling, "Discussion on a new concept of measuring instruments-Instrument Cloud," *China Measurement Test*, vol. 38, no. 2, pp. 1–5, 2012.

[27] V. Rajesh, O. Pandithurai, and S. Mageshkumar, "Wireless sensor node data on cloud," in *Proc. IEEE Int. Conf. Commun. Control Comput. Technol.*, 2010, pp. 476–481.

[28] Y. Takabe, K. Matsumoto, M. Yamagiwa, and M. Uehara, "Proposed sensor network for living environments using cloud computing," in *Proc. 15th Int. Conf. Netw.-Based Inf. Syst.*, 2012, pp. 838–843.

[29] G. Zhongwen, L. Chao, F. Yuan, and H. Feng, "CCSA: A cloud computing service architecture for sensor networks," in *Proc. Int. Conf. Cloud Service Comput.*, 2012, pp. 25–31.

[30] M. Armbrust, et al., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[31] H. He, Z. Wei, and S. Huang, "Future trend of integrating instrumentation into the cloud," in *Proc. Int. Workshop Cloud Comput. Inf. Secur.*, vol. 52, no. 2, pp. 359–365, 2013.

[32] T. Hirofuchi, E. Kawai, K. Fujikawa, and H. Sunahara, "USB/IP: A transparent device sharing technology over IP network," *IPSJ Dig. Courier*, vol. 1, pp. 394–406, 2005.

[33] P. Kalagiakos and P. Karampelas, "Cloud computing learning," in *Proc. 5th Int. Conf. Appl. Inf. Commun. Technol.*, 2011, pp. 1–4.

[34] A. Toshniwal, et al., "Storm@twitter," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 147–156.

[35] H. Karimipour and V. Dinavahi, "Parallel domain decomposition based distributed state estimation for large-scale power systems," *IEEE Trans. Ind. Appl.*, vol. 16, no. 2, pp. 1–6, Mar.-Apr. 2016.

[36] M. Shahidehpour and Y. Wang, *Communication and Control in Electric Power Systems: Applications of Parallel and Distributed Processing*, 1 ed. Hoboken, NJ, USA: Wiley, 2004, pp. 235–249.

[37] Y. Erkeng, *Power System State Estimation*. Beijing, China: Hydroelectric Power Publishing Company, 1985.

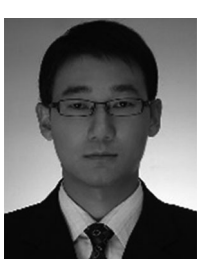
[38] H. Karimipour and V. Dinavahi, "Parallel domain decomposition based distributed state estimation for large-scale power systems," in *Proc. IEEE/IAS 51st Ind. Commercial Power Syst. Techn. Conf.*, 2015, pp. 1–5.

[39] P. Sanders and C. Schulz, "Think locally, act globally: Highly balanced graph partitioning," in *Experimental Algorithms*, 1 ed. Berlin, Germany: Springer, 2013, pp. 164–175.

[40] J. Zawodny, "Redis: Lightweight key/value store that goes the extra mile," *Linux Magazine*, Aug. 2009. [Online]. Available: www.linux-mag.com/id/7496/

[41] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[42] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Edition MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.



Hengjing He received the master's degree from Shanghai Jiao Tong University, China, in 2011. He is now working toward the PhD degree at Tsinghua University. His research interests include computerized measurement and instrumentation, cloud based instrumentation and real-time big on data processing in measurement systems.

Wei Zhao received the PhD degree from Moscow Energy Institute, Russia, in 1991. He is now a professor with Tsinghua University. His research areas include electromagnetic measurement, virtual instrumentation, networked instrumentation system, cloud based instrumentation.

Songling Huang received the PhD degree from Tsinghua University, in 2001. Currently, he is a professor in the Department of Electrical Engineering, Tsinghua University. His current research interests include electromagnetic measurement and nondestructive evaluation.

Geoffrey C. Fox received the PhD degree in theoretical physics from Cambridge University, in 1967 and is now the associate dean for Research and Graduate Studies at the School of Informatics and Computing, Indiana University, Bloomington, and professor of computer science, informatics, and physics with Indiana University where he is director of the Community Grids Laboratory. His research interests include data science, parallel and distributed computing.

Qing Wang received the PhD degree from the De Montfort University, United Kingdom, in 2001. She is now a lecturer in the School of Engineering and Computing Sciences, Durham University. Her research interests include electronic instruments and measurement, computer simulation, and advanced manufacturing technology. She is a charted engineer (CEng), a senior member of the IEEE (SMIEEE), a member of the IMechE (MIMechE), a member of the ASME (MASME) and a fellow of the Higher Education Academy (FHEA).

SSL: A *S*urrogate-Based Method for Large-Scale *S*tatistical *Latency* Measurement

Xu Zhang, Hao Yin, Dapeng Oliver Wu, *Fellow, IEEE*, Haojun Huang, Geyong Min, and Ying Zhang

Abstract—Understanding the statistical latency between two groups of hosts in a period of time is of great significance to a wide variety of Internet applications and services, such as Service-Level Agreement (SLA) compliance monitoring and Virtual Network Function (VNF) placement. However, direct latency measurement methods are not always applicable to large-scale situations while the existing indirect methods often incur extra deployment costs or security problems. To address this challenge, we design an indirect method based on widely-distributed clients called *SSL* (Surrogate-based method for large-scale Statistical Latency measurement). *SSL* estimates the latency between two arbitrary hosts using the measured latencies from several selected clients near one end host, which are called the host's surrogates, to the other end host. To overcome the limited capacity of the volatile clients with unstable CPU, memory, and bandwidth resources, we propose an innovative two-step measurement task assignment mechanism for *SSL* that can achieve high accuracy measurement results while satisfying the resource constraints simultaneously. Moreover, *SSL* adopts a sampling technique to reduce the overhead in large-scale measurements, and a resampling technique to determine the confidence interval. Simulation experiments show that *SSL* can achieve more than 90 percent accuracy in most situations with 10 percent client density and 15 percent sampling rate.

Index Terms—Statistical latency, latency measurement, resampling, task assignment, surrogate

1 Introduction

The ability to estimate the real-time round-trip latency between any pair of hosts on the Internet is important for a variety of applications and services, such as candidate service selection in service compositions [1], server assignment in content delivery [2], and path construction in overlay routing [3]. It is worth noting that for some applications it is not a must to compute the precise latency between any pair of hosts on the Internet [4], [5]. Instead, accurately estimated latency statistics of a set of paths can benefit many applications, such as SLA compliance monitoring [6], Content Delivery Network (CDN) site provisioning [7], service recommendation for web services [8], and VNF placement [9]. For example, for replica placement applications, it does not require a precise value of the latency between a particular pair of hosts (e.g., Alice in New York and Bob in San Francisco), but rather, the average latencies or 95-percentile latencies between hosts in these two cities.

In this paper, we propose a new type of Internet scale latency measurement called *statistical latency measurement*, which aims at accurately performing the *statistical estimate* capturing the *statistical value* of the latencies between two groups of hosts in a given period of time. Note that this new measurement method is not a replacement to the real-time exact measurement, but a complementary method for certain applications. In order to conduct the statistical latency measurement, many factors should be taken into consideration, for example, the measurement overhead, measurement accuracy, investment budgets, and the influence to the Internet, especially in large-scale scenarios.

The existing latency measurement methods can be classified into two categories: direct methods and indirect methods. Direct methods, which send measurement packages, e.g., Internet Control Message Protocol (ICMP) ping packages directly from target hosts, are not feasible as they need control on the target hosts. Apart from the direct measurement methods, there has been a rich literature on indirect measurement for the end-to-end real-time latency on the Internet [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]. However, none of them satisfies the requirements of statistical latency measurement due to the inaccuracy of predictions [11], the high extra costs [10] incurred by deploying numerous servers, the high measurement overhead [21], or even unfriendliness to the Internet [12] by bringing about security issues [22], [23].

This paper introduces a lightweight, scalable and accurate method for statistical latency measurement on the Internet. For many network entities such as Google, Microsoft and BitTorrent, they always have hundreds of thousands of users widely-distributed over the Internet, with devices

- *X. Zhang is with the School of Electronic Science and Engineering, Nanjing University, Jiangsu, China, 210023. E-mail: xzhang17@nju.edu.cn.*
- *H. Yin is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: h-yin@mail.tsinghua.edu.cn.*
- *D.O. Wu is with the Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL 32611. E-mail: wu@ece.ufl.edu.*
- *H. Huang is with the Department of Communication Engineering, Wuhan University, Wuhan 430072, China. E-mail: hhj0704@hotmail.com.*
- *G. Min is with the Department of Mathematics and Computer Science, University of Exeter, Exeter EX4, United Kingdom. E-mail: g.min@exeter.ac.uk.*
- *Y. Zhang is with the Networking and Mobility Lab, Hewlett Packard Labs, Palo Alto, CA 94304. E-mail: ying.zhang13@hpe.com.*

Manuscript received 19 Mar. 2016; revised 14 June 2017; accepted 18 July 2017. Date of publication 3 Aug. 2017; date of current version 13 Oct. 2020. (Corresponding authors: Xu Zhang and Hao Yin.)
Digital Object Identifier no. 10.1109/TSC.2017.2735409

installed with their specially-developed software[1] or plug-ins.[2] These devices are also called clients, which may include desktop, tablet, notebook computer and even smart phones.[3] For example, a notebook computer installed with BitTorrent software can measure the Internet in the background. If these entities themselves want to provide decision support for their services or even to a third-party company, they can take advantage of the clients to measure the Internet. Our key idea is to take advantage of these clients to measure the Internet, instead of deploying numerous extra controllable nodes in the network. However, there exist many challenges in the exploitation of these clients because they are always instable and have limited resources, such as bandwidths, CPU, and memory.

In order to measure the statistical latency in large-scale networks, this paper makes the following contributions.

1) We propose a surrogate-based method called *SSL* (*S*urrogate-based method for large-scale *S*tatistical *Latency* measurement) that takes advantage of the free but volatile clients to conduct large-scale statistical latency measurement. Since it does not deploy extra servers in the network, *SSL* does not incur extra deployment cost but can achieve high measurement accuracy.
2) We introduce space diversity into *SSL* to improve the measurement accuracy. When measuring the latency between two hosts, *SSL* selects the q closest active clients to one of the two hosts, which are called the host's surrogates, measures the distance from the surrogates to the other host, and then predicts the latency between the two hosts. Moreover, this method can overcome the problem of asymmetric routing [16] in the Internet.
3) We adopt sampling and resampling techniques to reduce the measurement overhead and to obtain statistical estimates. *SSL* samples the end-host pairs needed to be measured, measures them using the aforementioned latency prediction methods, and resamples the samples to determine the confidence interval and other statistical parameters.
4) We take advantage of a two-stage mechanism to assign measurement tasks to the clients, aiming at obtaining the measurement results as accurate as possible without affecting users' Quality of Experience (QoE). Different from deployed servers, clients are always volatile and limited with various resources, such as network bandwidths, CPU, and memory. Furthermore, different clients always have different resource capacities available. With the measurement task assignment mechanism, *SSL* can achieve high accuracy measurement results while satisfying the resource constraints simultaneously.
5) Last but not least, we evaluate *SSL* in a router-level network topology, and the results reveal the promising performance of *SSL*. *SSL* can achieve less than 10 percent error in most situations with 10 percent client density and 15 percent sampling rate. We also explore the design choices of *SSL*, including the optimal number of the surrogates and the way to use the latencies measured by them.

The rest of this paper is organized as follows. Section 2 makes clear the potential and challenges when taking advantage of clients to measure the Internet. Section 3 formulates the statistical measurement problem from spatial dimension and temporal dimension. Section 4 presents the key components of *SSL*, including the method to predict the latency between two arbitrary hosts, the way to reduce the measurement overhead at scale and the measurement task assignment mechanism to satisfy the limited source constraints. The proposed method is evaluated in Section 5, and the related work is reviewed in Section 6. Finally, Section 7 summarizes the paper.

2 Potential and Challenges

Predicting the latency between two arbitrary hosts at scale should take various factors into consideration, such as the prediction accuracy, investment budgets, measurement overheads, scalability, and effects on the Internet. In this section, we highlight the potential and challenges when using the widely-distributed clients to measure the Internet.

Apart from the deployed servers, another important component of the Internet is the widely-distributed end-hosts. If we can take advantage of the end-hosts to measure the Internet, a more accurate prediction can be expected. Fortunately, end-hosts are likely to participate in the measurement if they install specially-developed software, such as Bittorrent plug-ins[4] or specific agent.[5] We call these end-hosts installed with specific software as clients. Another advantage is that these clients can be used free of charge, avoiding extra deployment cost. However, there exist many challenges to use the widely-distributed clients. These challenges are:

1) *The volatility of the clients:* the clients come and go from time to time, thus not available all the time. Only when users' devices are online, these clients can be used for measurement. However, the clients are users' personal computers or even mobile devices in many cases and users would switch the clients' network status according to their behavior habit. For example, a user would turn off the device after work or move the device from one place to another place. The instability of clients would bring great challenges if they are adopted to measure the Internet.
2) *The limited resource of the clients:* as the clients are installed on users' devices, the usage of the clients should not consume too much resource to degrade users' experience. For example, the measurement program running in the background should not consume too much CPU, memory, bandwidth. In the mobile case, energy saving should also be taken into consideration [24].
3) *The diversity of the clients:* users' devices and their network condition may differ from each other greatly.

1. http://www.bittorrent.com/
2. https://www.httpwatch.com/
3. http://www.measurementlab.net/tools/mobiperf
4. http://www.aqualab.cs.northwestern.edu/projects/115-dasu-isp-characterization-from-the-network-edge
5. http://www.netdimes.org/new/?q=node/54

Different devices have different limited resource. For example, mobile clients with wireless access to the Internet have much rigorous limitation on the available bandwidth and energy saving.

3 Problem Formulation

In this section, we formulate the problem of *statistical latency measurement* and describe its general steps.

The statistical latency measurement is the statistical estimate of the latencies between two groups of end hosts in a given period. It provides information to infer the connectivity status between different network entities. The connectivity status can be used to improve the performance of various applications, such as site provisioning for CDNs, and CDN selection. In what follows, we will define this problem formally from both temporal and spatial dimensions.

Spatial Dimension. Given two host sets $X : [x_1, \ldots, x_n]$ and $Y : [y_1, \ldots, y_m]$, the latency matrix between X and Y at time point t is

$$D^t_{\overrightarrow{X,Y}} = \begin{bmatrix} d^t_{\overrightarrow{x_1,y_1}} & \cdots & d^t_{\overrightarrow{x_1,y_m}} \\ \vdots & \ddots & \vdots \\ d^t_{\overrightarrow{x_n,y_1}} & \cdots & d^t_{\overrightarrow{x_n,y_m}} \end{bmatrix}, \tag{1}$$

where $d^t_{\overrightarrow{x_i,y_j}}$ represents the latency from host x_i in X to host y_j in Y at time point t. $D^t_{\overrightarrow{X,Y}}$ reflects the status of the connectivity from host sets X to Y at time point t. Then the statistical latency from X to Y at time point t can be defined as follows:

$$d^t_{\overrightarrow{X,Y}} = \frac{1}{mn} \sum_{x \in X, y \in Y} d^t_{\overrightarrow{x,y}}. \tag{2}$$

Temporal Dimension. Now, we extend the statistical latency in temporal dimensions. Given two host sets X and Y, the statistical latency from X to Y over the time period T is

$$d^T_{\overrightarrow{X,Y}} = \frac{1}{T} \int_0^T d^t_{\overrightarrow{X,Y}} \mathrm{d}t. \tag{3}$$

However, we cannot measure every host pair continuously over the time period T in the large-scale Internet. Usually we collect multiple latencies at $[t_1, \ldots, t_r]$ over time T and construct the latency matrix, that is, $D^T_{\overrightarrow{X,Y}}$, just like a series of snapshots to the Internet

$$D^T_{\overrightarrow{X,Y}} = \left[D^{t_1}_{\overrightarrow{X,Y}}, \ldots, D^{t_r}_{\overrightarrow{X,Y}} \right], \tag{4}$$

where T is discrete, that is, $T = [t_1, \ldots, t_r]$. Then $d^T_{\overrightarrow{X,Y}}$ can be rewrited as follows:

$$d^T_{\overrightarrow{X,Y}} = \frac{1}{mnr} \sum_{t \in T} \sum_{x \in X, y \in Y} d^t_{\overrightarrow{x,y}}. \tag{5}$$

Supposed that the Internet status stays relatively stationary within a short period, which is the assumption for all the existing measurement works. Latencies are treated to be measured at the same time point if they are measured within the period. To this end, we use the discrete time model because we cannot measure every host pair continuously over the time period T in the large-scale Internet. Of course, the discrete time model would reduce the measurement accuracy. However, with the development of the Internet infrastructure, including the rapid increase of computing capacity, the memory capacity and the bandwidth capacity, the accuracy would be improved in the future.

For site provisioning, operators can use *SSL* to evaluate the performance of a given edge server across a time period T. Under this circumstance, set X would be the given server x, set Y be the users who request services from x, the statistical latency $d^T_{\overrightarrow{X,Y}}$ represents the overall latency performance when x acts as a server and provides service to hosts in set Y across time period T.

For CDN selection, *SSL* can be used to infer the connectivity status between the edge servers of a CDN and its users over a time period, that is, a CDN's overall latency in distributing contents. In this case, we can measure the statistical latency between each edge server and its allocated users, and compute the overall average latency when users fetch content from the CDN.

Specially, if set X and Y both consist of only one host and the time period T contains only a time point t, $D^T_{\overrightarrow{X,Y}}$ would be the real-time latency between the two hosts at time point t in the Internet.

In general, the first step to complete a statistical latency measurement is to measure the latency between any given two hosts, that is, filling in the latency matrix $D^T_{\overrightarrow{X,Y}}$ or $D^t_{\overrightarrow{X,Y}}$. After that, the next step is to infer the overall connectivity status between host set X and set Y. However, how to measure the latency at scale with adequate accuracy is a big challenge.

4 Methodology

In this section, we highlight three key techniques of *SSL*. Table 1 lists the main terminologies used in the description.

4.1 End-to-End Latency Prediction

In order to infer the statistical latency between two host sets, *SSL* should have the ability to measure the end-to-end latency between two hosts, which can also be used for real-time applications, such as peer selection in P2P systems, server selection in online game and candidate service selection in service compositions.

SSL takes advantage of the clients in the Internet, which are always users' devices and widely distributed. As illustrated in Fig. 1, an intuitive solution is to select the closest client to one host (host x), and use the latency from the client to the other host (host y) to estimate the latency from host x to host y. In an extreme situation where host x is a client itself, the estimated latency is exactly the measured latency. Furthermore, this solution can also overcome the challenges of asymmetric paths in the Internet [16].

Based on the above solution, *SSL* introduces *space diversity* to reduce occasional errors. As shown in Fig. 2, to estimate the latency ($d^t_{\overrightarrow{x,y}}$) from host x to host y at time point t, *SSL* selects the q closest clients to host x, for example, $\{z_1, \ldots, z_q\}$. Then it estimates $d^t_{\overrightarrow{x,y}}$ according to the latency

TABLE 1
The Main Terminologies Used in the Description of *SSL*'s Methodology

Notation	Semantics
X	$\{x_1, \ldots, x_n\}$, n is the number of hosts in X, and i is the index.
Y	$\{y_1, \ldots, y_m\}$, m is the number of hosts in Y, and j is the index.
T	$\{t_1, \ldots, t_r\}$, r is the number of time points in T, and v is the index.
Z	$\{z_1, \ldots, z_l\}$, l is the number of clients in Z, and k is the index.
P	Measured tasks in order to infer the statistical latency from X to Y across the time period T, and $P = \{d^{t_v}_{\overrightarrow{x_i, y_j}}\}$, where $x_i \in X$, $y_j \in Y$, $t_v \in T$, and $\lvert P\rvert \leq mnr$
$\overline{d_P}$	Mean of the latencies in P
$\overline{d^*}$	Resampling mean of $\overline{d_P}$
$se(\overline{d})$	Standard errors of $\overline{d_P}$
CI	$[\overline{d^*_L}, \overline{d^*_R}]$, the $(1-\beta)$ confidence interval of $\overline{d_P}$.
q	Maximum number of surrogates to conduct a task.
$d^t_{\overrightarrow{a,b}}$	Latency from entity a to entity b at time t.
c^t_{ik}	Capacity of the client z_k reserved to x_i at time point t, $1 \leq k \leq l$.
g^t_i	Number of tasks which are originated from x_i among the measurement tasks P^t.
ω^t_{ik}	$\omega^t_{ik} = 1$ if z_k is selected to be a surrogate of x_i, $\omega^t_{ik} = 0$ otherwise.
$\mathfrak{d}_{ik}$	Distance between x_i and z_k when selecting the surrogate for a host.
$\mathfrak{d}_{max}$	Maximum distance between a host and its surrogate in a distance metric space when selecting the surrogates.
r^t_i	Number of tasks conducted by the surrogates of x_i at time point t

$d^t_{\overrightarrow{z_k, y}}$ from the client z_k to y, where $1 \leq k \leq q$. The selected clients are called x's *surrogates*.

If a client z is selected as a surrogate of host x, z should have the following properties to guarantee the representativeness to x:

- *Near Neighbor*: z should be within $\mathfrak{d}_{max}$ to x in a distance metric space, which can be computed based on the longest matching prefix, the hop number, the geographic distance[6] between their IPs, or even the combination of the factors. Moreover, the class of the client z's IP and the class of host x's IP should also be the same.
- *Same Autonomous System (AS)*: z should be within the same AS[7] with x, sharing the common routing policy to the Internet.

We determine whether z can be selected as a surrogate of host x based on the above criterion due to two reasons: (1) the above criterion is lightweight, especially compared with the measurement overhead, that is, SSD conducts at most q measurements for each measurement task. It's critical in large-scale network. (2) SSD's accuracy is enough based on the above criterion. First, the density of clients is expected to be much higher than the density of open recursive DNS in the Internet. Second, the criterion to select clients is much more tight than King's criterion of selecting the open recursive DNS. Last but not least, SSD introduces space diversity to reduce occasional errors. From this perspective, ***SSL*** has much higher prediction accuracy than King, whose accuracy has been verified in many situations [25]. However, King is not suitable to the statistical latency measurement any longer due to the security issues.

6. The geographic distance between two IPs can be computed based on their longitude and latitude, for example, https://www.db-ip.com/db/.

7. The AS number of an IP can be looked up via online API, for example, https://www.ultratools.com/tools/asnInfo.

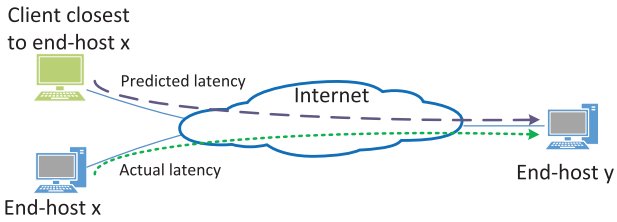


Fig. 1. One intuitive solution: Predicting the latency from end-host x to end host y using the latency from the closest client of x to y.

Of course, there may exist many other factors or other ways that could further improve z's representativeness to x, such as methods based on x and z's network latencies to several landmarks in the Internet. ***SSL*** is flexible to plugin them.

Based on the latencies $d^t_{\overrightarrow{z_k, y}}$ from the surrogates $z_k \in \{z_1, \ldots, z_q\}$ to y, ***SSL*** can choose several optional ways to estimate $d^t_{\overrightarrow{x,y}}$. The first one is to estimate $d^t_{\overrightarrow{x,y}}$ using the *average* of $d^t_{\overrightarrow{z_k, y}}$, where $1 \leq k \leq q$

$$d^t_{\overrightarrow{x,y}} = \frac{1}{q}\sum_{k=1}^{q} d^t_{\overrightarrow{z_k, y}}. \tag{6}$$

Second, ***SSL*** can also adopt the *weighted average* to consider that different surrogates have different distances to host x. In this situation, the closer a surrogate z_k to host x, the higher weight its latency $d^t_{\overrightarrow{z_k, y}}$ to y has

$$d^t_{\overrightarrow{x,y}} = \sum_{k=1}^{q} \alpha_k d^t_{\overrightarrow{z_k, y}}, \tag{7}$$

where α_k is inversely proportional to $d^t_{\overrightarrow{z_k, x}}$ and $\sum_k \alpha_k = 1$.

Moreover, ***SSL*** can also introduce *trimmed average* into latency prediction. In detail, ***SSL*** can first sort $\{d^t_{\overrightarrow{z_1, y}}, \ldots, d^t_{\overrightarrow{z_k, y}}, \ldots, d^t_{\overrightarrow{z_q, y}}\}$ $(1 \leq k \leq q)$ into $\{d^t_{\overrightarrow{z^1, y}}, \ldots, d^t_{\overrightarrow{z^k, y}}, \ldots,$

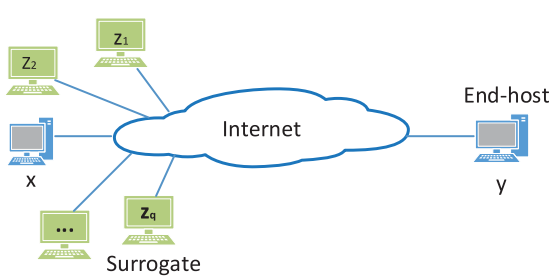


Fig. 2. Predicting the latency between two end-hosts by introducing space diversity to reduce occasional errors.

$d^t_{\overrightarrow{\ }}\}$ $(1 \le k \le q)$ according to the ascending order, where $d^{k^q,y}_{\overrightarrow{z^k,y}}$ is the k_{th} smallest value in $\{d^t_{\overrightarrow{z_1,y}}, \ldots, d^t_{\overrightarrow{z_k,y}}, \ldots, d^t_{\overrightarrow{z_q,y}}\}$ $(1 \le k \le q)$. Then the first p-quantile values and the last p-quantile values in $\{d^t_{\overrightarrow{z^1,y}}, \ldots, d^t_{\overrightarrow{z^k,y}}, \ldots, d^t_{\overrightarrow{z^q,y}}\}$ $(1 \le k \le q)$ are removed from the sequence, and the mean of the remaining elements in the sequence is used to estimate $d^t_{\overrightarrow{x,y}}$

$$d^t_{\overrightarrow{x,y}} = \frac{1}{q_p}\sum_{k=1}^{q} d^t_{\overrightarrow{z^k,y}} I\{q * p\% \le k \le q * (1 - p\%)\}, \quad (8)$$

where $I\{x\}$ is the indicative function, which equals 1 if x is true, otherwise equals 0; and q_p is the number of elements in the trimmed sequence,

$$q_p = \sum_{k=1}^{q} I\{q * p\% \le j \le q * (1 - p\%)\}. \quad (9)$$

Similarly, ***SSL*** can introduce different weights into α-trimmed average easily. Moreover, if there exist other ways to estimate $d^t_{\overrightarrow{x,y}}$ based on $d^t_{\overrightarrow{z_k,y}}$ $(1 \le k \le q)$, ***SSL*** is flexible to plugin it.

4.2 Overhead Reduction at Scale

When inferring the statistical latency from host set $X = \{x_1, \ldots, x_n\}$ to host set $Y = \{y_1, \ldots, y_m\}$ over time $T = \{t_1, \ldots, t_r\}$, the overall measurement overhead is $o(mnr)$ if measuring all the host pairs between X and Y. If the host number in each host set is small and conducting the latency prediction measurements for $o(mnr)$ times in T is acceptable, measuring the universal set (all the elements in $D^T_{\overrightarrow{X,Y}}$) can be adopted. However, for the large scale case, the brute force method is too expensive to be feasible.

To overcome the challenge, ***SSL*** randomly samples the universal set and takes the samples to infer $D^T_{\overrightarrow{X,Y}}$'s statistical information. These sampled elements which should be measured are denoted as $P = \{d^{t_v}_{\overrightarrow{x_i,y_j}}\}$, where $x_i \in X$, $y_j \in Y$, $t_v \in T$, and $|P| < mnr$. We call P as ***SSL***'s measurement tasks for the statistical latency from X to Y over time period T.

Without loss of generality, we denote the sampled elements with $\{d^{t^1}_{\overrightarrow{x^1,y^1}}, \ldots, d^{t^b}_{\overrightarrow{x^b,y^b}}, \ldots, d^{t^s}_{\overrightarrow{x^s,y^s}}\}$, where $x^b \in X$, $y^b \in Y$, $t^b \in T$, and $1 \le b \le s < mnr$. As the Internet status stays relatively stationary within a short period, we can use the average of P to represent $d^T_{\overrightarrow{X,Y}}$ if s is large enough

$$\overline{d_P} = \frac{1}{s}\sum_{b=1}^{s} d^{t^b}_{\overrightarrow{x^b,y^b}} \approx d^T_{\overrightarrow{X,Y}}. \quad (10)$$

In order to assess the accuracy of the sampling method, ***SSL*** introduces a kind of resampling technique, that is, Bootstrap [26], [27] into its design. The basic idea is that the observed sample P contains valuable information about the variability about the population of the latencies from X to Y in absence of any other information. The procedures to obtain the resampling mean $\overline{d^*}$, the standard error $se(\overline{d_P})$, the $(1-\beta)$ confidence interval $CI = [\overline{d_L}, \overline{d_R}]$ are described by Algorithm 1. Note that the expectation of the resampling average $\overline{d^*}$ is equal to $\overline{d_P}$.

Algorithm 1. Resampling(P, B, β)

1: [Input] P: the sampled elements $\{d^{t^1}_{\overrightarrow{x^1,y^1}}, \ldots, d^{t^s}_{\overrightarrow{x^s,y^s}}\}$.
2: [Input] B: the time to resample P.
3: [Input] β: the parameter for $(1-\beta)$ confidence interval.
4: [Output] $\overline{d^*}$: the resampling average.
5: [Output] $se(\overline{d_P})$: the standard error.
6: [Output] CI: the $(1-\beta)$ confidence interval $[\overline{d_L}, \overline{d_R}]$.
7: **for** $b = 0$; $b < B$; b++ **do**
8: $P^{*(b)} \leftarrow \{d^{t^{*1}}_{\overrightarrow{x^{*1},y^{*1}}}, \ldots, d^{t^{*c}}_{\overrightarrow{x^{*c},y^{*c}}}, \ldots, d^{t^{*s}}_{\overrightarrow{x^{*s},y^{*s}}}\}$, where $d^{t^{*c}}_{\overrightarrow{x^{*c},y^{*c}}}$ is sampled with replacement from P;
9: // Compute the average of $P^{*(b)}$.
10: $\overline{d^{*(b)}} \leftarrow \frac{1}{s}\sum_{c=1}^{s} d^{t^{*c}}_{\overrightarrow{x^{*c},y^{*c}}}$;
11: $\overline{d^*} \leftarrow \frac{1}{B}\sum_{b=1}^{B} \overline{d^{*(b)}}$;
12: $se(\overline{d_P}) \leftarrow \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\overline{d^{*(b)}} - \overline{d^*})}$;
13: //Compute the lower bound of $\overline{d_P}$'s $(1-\beta)$confidence interval
14: $\overline{d_L} \leftarrow \overline{d^*_L}$, where $\overline{d^*_L}$ satisfies that $\mathbb{P}(\overline{d_P} \le \overline{d^*_L}) = \frac{1}{B}\sum_{b=1}^{B} I\{\overline{d^{*(b)}} \le \overline{d^*_L}\} \approx \frac{1}{2}\beta$;
15: //Compute the upper bound of $\overline{d_P}$'s $(1-\beta)$confidence interval
16: $\overline{d_R} \leftarrow \overline{d^*_R}$, where $\overline{d^*_R}$ satisfies that $\mathbb{P}(\overline{d_P} \ge \overline{d^*_R}) = \frac{1}{B}\sum_{b=1}^{B} I\{\overline{d^{*(b)}} \ge \overline{d^*_R}\} \approx \frac{1}{2}\beta$;
17: return $\overline{d^*}, se(\overline{d_P}), CI = [\overline{d_L}, \overline{d_R}]$;

Lemma 4.1. *The expectation of the resampling average $\overline{d^*}$ is equal to $\overline{d_P}$.*

Proof. As Algorithm 1 shows, for each sampled sequence in $P^{*(b)}$, its average is computed as follows,

$$\overline{d^{*(b)}} = \frac{1}{s}\sum_{c=1}^{s} d^{t^{*c}}_{\overrightarrow{x^{*c},y^{*c}}} = \frac{1}{s}\sum_{c=1}^{s} N_c d^{t^c}_{\overrightarrow{x^c,y^c}},$$

where N_c is the observed frequency of $d^{t^c}_{\overrightarrow{x^c,y^c}}$ in the sequence and $c \in [1, s]$. As $d^{t^c}_{\overrightarrow{x^c,y^c}}$ is resampled with probability $\frac{1}{s}$ from P and ***SSL*** has resampled P with replacement for s times, we have

$$\mathbb{E}(N_c) = s * \frac{1}{s} = 1.$$

It follows that

$$\begin{aligned}\mathbb{E}(\overline{d^{*(b)}}) &= \mathbb{E}\left(\frac{1}{s}\sum_{c=1}^{s} N_c d^{t^c}_{\overrightarrow{x^c,y^c}}\right)\\ &= \frac{1}{s}\sum_{c=1}^{s} d^{t^c}_{\overrightarrow{x^c,y^c}}\\ &= \overline{d_P}.\end{aligned}$$

In addition, the resampling average is given by

$$\overline{d^*} = \frac{1}{B}\sum_{b=1}^{B} \overline{d^{*(b)}},$$

thus $\mathbb{E}(\overline{d^*}) = \overline{d_P}$. □

4.3 Measurement Task Assignment Mechanism

After determining the measurement tasks, *SSL* should appoint usable clients to conduct them. On one hand, the clients are usually users' devices and to be on and off from time to time. On the other hand, resources on these clients are always limited, such as bandwidths, CPU, memory etc., leading to a limited capacity available. Considering this, not every task can be conducted by the most appropriate clients. To address this problem, *SSL* designs a specific task assignment mechanism to achieve a better inferring accuracy. The overall process of the mechanism can be divided into two steps.

First step: maximize the number of tasks, each of which can be completed by at least one surrogate. The problem is abbreviated as (**MTN**).

The measurement tasks $P = \{d^{t_v}_{\overrightarrow{x_i,y_j}}\}$, where $x_i \in X$, $y_j \in Y$, $t_v \in T$, and $|P| < mnr$. In order to complete the tasks, *SSL* divides the measurement tasks according to the time point, gets the tasks for each time point $P^t = \{d^{t_v}_{\overrightarrow{x_i,y_j}} | d^{t_v}_{\overrightarrow{x_i,y_j}} \in P \& t^v = t\}$, and takes advantage of the client set $Z = \{z_1, \ldots, z_k, \ldots, z_l\}$ to conduct the tasks.

Let g^t_i be the number of tasks which are originated from x_i among the measurement tasks P^t. And for the client z_k, it can conduct c^t_k measurements simultaneously at time point t.

SSL assigns the measurement tasks dynamically for each time point t according to the measurement task P^t and the clients' capacities $\{c^t_k | 1 \le k \le l\}$. If a client z_k is offline at the time point t, the number of measurements it can conduct at the time point t is 0, that is, $c^t_k = 0$; if z_k is online, it should report its status with heartbeat messages, thus the number of measurements c^t_k it can conduct at the time point t can be obtained, that is, we have different value of c^t_k at different time points.

Suppose ω^t_{ik} is a binary indicator, and $\omega^t_{ik} = 1$ if z_k is selected to be a surrogate of x_i at time point t, and $\omega^t_{ik} = 0$ otherwise

$$\omega^t_{ik} \in \{0, 1\}. \tag{11}$$

If $\omega^t_{ik} = 1$, c^t_{ik} denote the capacity reserved to x_i from z_k at time point t.

Due to the capacity limitation for a given client, *SSL* should not appoint it more than c^t_k tasks

$$\sum_i \omega^t_{ik} c^t_{ik} \le c^t_k. \tag{12}$$

Let $\mathfrak{d}_{ik}$ denote the distance between the host x_i and the client z_k in the distance metric space when selecting a surrogate. The surrogate of a host should be within $\mathfrak{d}_{max}$ to it. Then we have

$$\omega^t_{ik} \mathfrak{d}_{ik} \le \mathfrak{d}_{max}. \tag{13}$$

If $\mathfrak{d}_{ik} > \mathfrak{d}_{max}$, ω^t_{ik} should be equal to 0, that is, z_k could not be appointed to be a surrogate of x_i; otherwise, ω^t_{ik} can be equal to 1 or 0, that is, *SSL* is free to assign z_k to be a surrogate of x_i or not.

Let r^t_i be the number of tasks which are conducted by the surrogates of x_i at time point t. As the maximum number of tasks originated from x_i that can be conducted is g^t_i, the relationship between r^t_i and ω^t_{ik} can be expressed by

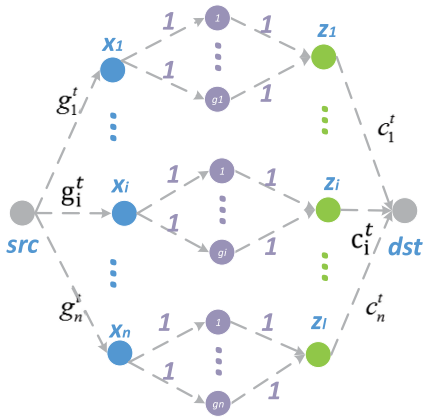


Fig. 3. Transform the MTN problem into a maximum flow problem.

$$r^t_i = \begin{cases} \sum_k \omega^t_{ik} c^t_{ik} & \text{if } \sum_k \omega^t_{ik} c^t_k < g^t_i \\ g^t_i & \text{if } \sum_k \omega^t_{ik} c^t_k \ge g^t_i, \end{cases} \tag{14}$$

where $\sum_k \omega^t_{ik}$ is the number of surrogates of host x_i at time point t, while $\sum_k \omega^t_{ik} c^t_k$ is the maximum capacity of x_i's surrogates.

In this step, we aim at maximizing the number of tasks, each of which is assigned to at least one surrogate. Therefore, the objective function is

$$\max. \sum_i r^t_i. \tag{15}$$

So MTN problem can be formulated into the following optimization problem:

$$\max. \sum_i r^t_i$$
$$\begin{cases} \omega^t_{ik} \in \{0, 1\} \\ \sum_i \omega^t_{ik} c^t_{ik} \le c^t_k \\ \omega^t_{ik} \mathfrak{d}_{ik} \le \mathfrak{d}_{max} \\ r^t_i = \begin{cases} \sum_k \omega^t_{ik} c^t_{ik} & \text{if } \sum_k \omega^t_{ik} c^t_{ik} < g^t_i \\ g^t_i & \text{if } \sum_k \omega^t_{ik} c^t_{ik} \ge g^t_i. \end{cases} \end{cases} \tag{16}$$

Lemma 4.2. *The MTN problem can be solved in polynomial time.*

Proof. We can transform the MTN problem into a maximum flow problem. As shown in Fig. 3, the capacity of the edge from the *src* node to the node x_i has the capacity of g^t_i, which corresponds to the number of the tasks related to host x_i in P^t. To make clear each individual task, g^t_i nodes are connected to the node x_i, with the edge capacity equal to 1. If a client z_k is within $\mathfrak{d}_{max}$ of a host x_i, an edge is originated from each task node of the node x_i to the node z_k. For each node z_k, it connects with the *dst* node with capacity of c^t_k, which is the maximum number of tasks the client z_k can conduct at time point t. As x_i has appeared for g^t_i times, the node x_i has g^t_i child nodes. And for a client z_k, it can conduct c^t_k measurements simultaneously.

To achieve the objective of maximizing the number of tasks, each of which is assigned to at least one surrogate, we just need to maximize the flow from the *src* node to the *dst* node. So the MTN problem can be solved in O $((|X|+|P^t|+|Z|)^2)$ [28]. □

In the first step, ***SSL*** maximizes the number of completed tasks and a task is regarded to be completed when it is conducted at least by one surrogate. However, if q surrogates conduct a task at the same time, the prediction accuracy would be improved. Considering this, ***SSL*** tries to maximize the number of surrogates in the second step under the condition that the number of conducted tasks is the largest. The problem is abbreviated as (**MPN**).

Suppose $\omega_{ik}^{t}{}'$ and $c_{ik}^{t}{}'$ are the output values of ω_{ik}^{t} and c_{ik}^{t} in the first step, respectively. In order to guarantee the number of conducted tasks, ***SSL*** retains the surrogate assignment for each task in the first step and tries to assign more surrogates to it

$$\omega_{ik}^{t} \geq \omega_{ik}^{t}{}' \tag{17}$$

$$c_{ik}^{t} \geq c_{ik}^{t}{}'. \tag{18}$$

Moreover, suppose that q different surrogates are enough for a task, and more surrogates would not improve the prediction accuracy, that is, ***SSL*** tries to assign up to q surrogates to each task. So we have

$$\sum_{k} \omega_{ik}^{t} c_{ik}^{t} \leq g_i^t q, \tag{19}$$

where $\sum_k \omega_{ik}^t c_k^t$ is the total capacity of x_i's surrogates while $g_i^t q$ is the maximum capacity needed for the tasks related to x_i.

For each task, a surrogate can conduct it for at most once. So the capacity c_{ik}^t assigned to x_i by z_k should not exceed the number of tasks related to x_i

$$c_{ik}^{t} \leq g_i^t. \tag{20}$$

In this step, we aim at maximizing the number of surrogates assigned to each task. Therefore, the objective function is

$$\max. \sum_{i}\sum_{k} \omega_{ik}^{t} c_{ik}^{t}. \tag{21}$$

So MPN problem can be formulated into the following optimization problem:

$$\max. \sum_{i}\sum_{k} \omega_{ik}^{t} c_{ik}^{t}$$
$$\begin{cases} \omega_{ik}^{t} \in \{0,1\} \\ \sum_i \omega_{ik}^{t} c_{ik}^{t} \leq c_k^t \\ \omega_{ik}^{t} \geq \omega_{ik}^{t}{}' \\ c_{ik}^{t} \geq c_{ik}^{t}{}' \\ c_{ik}^{t} \leq g_i^t \\ \sum_k \omega_{ik}^{t} c_{ik}^{t} \leq g_i^t q \\ \omega_{ik}^{t} \mathfrak{d}_{ik} \leq \mathfrak{d}_{max}. \end{cases} \tag{22}$$

Lemma 4.3. *The MPN problem can be solved in polynomial time.*

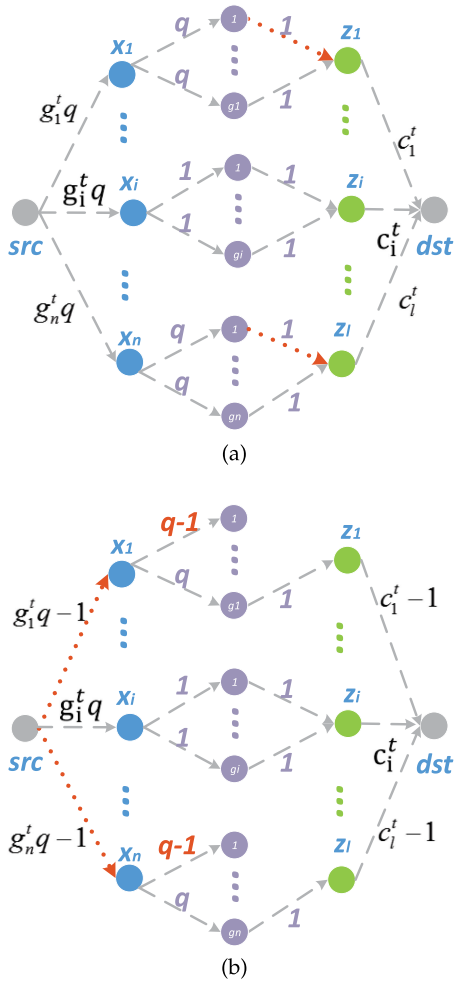


Fig. 4. Transform the MPN problem into a maximum flow problem.

Proof. We can also transform the MPN problem into a maximum flow problem. Similarly, we can first construct a graph similar to Fig. 3, as shown in Fig. 4a. As ***SSL*** tries to assign up to q different surrogates to each task, we revise the capacity of the edge that connects node x_i with its task nodes from 1 to q, and revise the capacity of the edge connected the *src* node with node x_i from g_i^t to $g_i^t q$. The dotted red line in Fig. 4a indicates that a client has been assigned to a task in the first step.

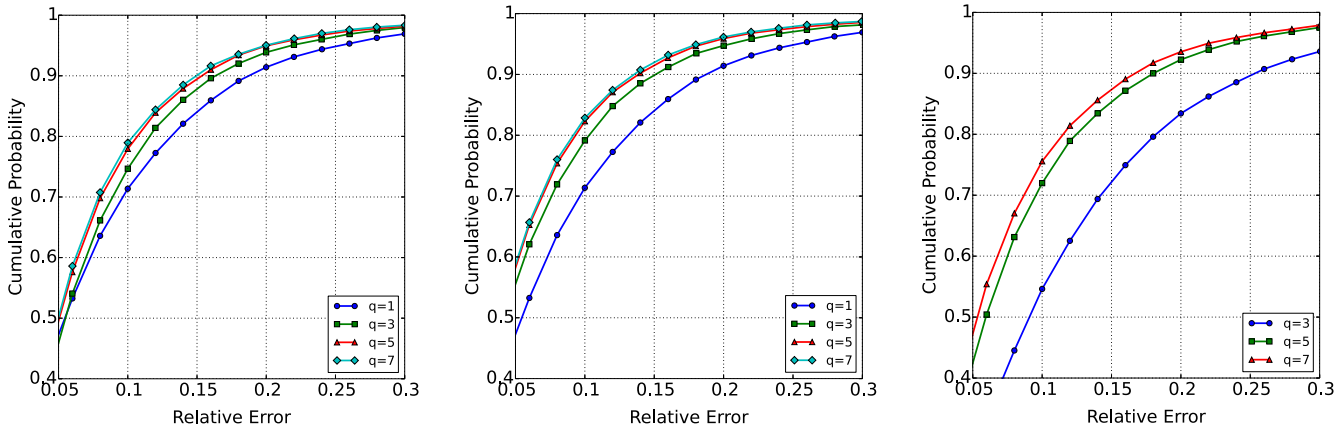
In order to guarantee the number of conducted tasks, we first remove the dotted red lines in Fig. 4a, as we would keep the assignment in solving the MPN problem. Next, the related edges should adjust their capacities accordingly, and we get the graph in Fig. 4b.

To achieve the objective of maximizing the number of surrogates assigned to each task, we just need to maximize the flow from the *src* node to the *dst* node in the graph and get the incremental assignment of the clients. So the overall MPN problem can be solved in O$((|X|+|P^t|+|Z|)^2)$. □

5 Evaluation

In this section, we evaluate ***SSL*** and explore ***SSL***'s design choices.

Experimental Setup. Compared with King, ***SSL*** has many advantages which can result to a much more accurate prediction. Unfortunately, we neither have enough clients distributed in the real-world Internet nor found existing data



(a) Average (b) Weighted Average (c) Trimmed Average

Fig. 5. The effect of the surrogate number for each task on *SSL*'s prediction accuracy.

sets to compare SSD with King. To evaluate *SSL* in absence of enough clients distributed in the real-world Internet, we generated various network topologies with different topology models and different network sizes using BRITE[8] and simulated *SSL* based on the generated topologies. For each network topology, we randomly chose nodes to act as the clients with given capacities, leaving others to act as the ordinary hosts in the Internet. The edge length between two nodes represents the latency between them. The latency between two arbitrary nodes is denoted by the length of the shortest path between them. To simulate the dynamic of the Internet, we make the length of each edge to vary within $\pm 10\%$ randomly of the value Brite generates.

Due to the space limit, we only shows the representative results on a router-level topology which contains up to 5,000 nodes. We have also found that the similar results were achieved for different topologies. In order to explore the impact of the density of clients on the prediction accuracy, we randomly chose 25-500 nodes to act as the clients. The capacity of each client has a positive correlation with link bandwidth connected to it.

Performance Metric. To weight how well *SSL* can predict the end-to-end latency, we use a metric called relative error, which is defined by the predicted latency between two hosts and the actual latency between them

$$\frac{|\text{predicted latency} - \text{actual latency}|}{\text{actual latency}}. \tag{23}$$

Thus the closer to zero the value of the relative error is, the more accurate the methodology can predict the end-to-end latency.

Impact of Surrogate Number on Prediction Accuracy. To measure the latency between two arbitrary hosts, *SSL* can select q surrogates and estimate the latency between the two hosts based on the latencies measured from the surrogates. In order to investigate the effects of the surrogate number q on *SSL*'s prediction accuracy, we changed the value of q from 1 to 7, and computed the relative errors for 10,000 randomly-sampled host pairs. The CDF for the relative errors when there exist 500 clients are depicted in Fig. 5.

In Fig. 5a, the curve with blue circle markers, the curve with green square makers, the curve with red triangle_up makers, and the curve with cyan diamond markers are the CDF curves for the relative errors when q is equal to 1, 3, 5, 7, respectively. As Fig. 5a illustrates, the prediction accuracy of *SSL* improves when the surrogate number q increases from 1 to 7. However, when q increases and exceeds 5, the margin of the improved prediction accuracy becomes much smaller, so that the CDF curves for the cases when q equals to 5 and 7 are nearly overlapping. The results are similar when the weighted mean and trimmed mean are adopted to estimate the latency, as shown in Figs. 5b and 5c. Considering this, we choose the maximum number of different surrogates for a task to be 5 in this case.

Design Choice on Latency Prediction Based on q Surrogates. After measuring the latency from q surrogates, *SSL* can have different choices to estimate the latency between the two related hosts, that is, the mean, the weighted mean, or the trimmed mean, as mentioned in Section 4.

Fig. 6 compares the prediction accuracy under the three different methods when the surrogate number q is equal to 5. In the figure, the curve with blue circle markers, the curve with green square makers, and the curve with red triangle_up makers are the CDF curves for relative errors

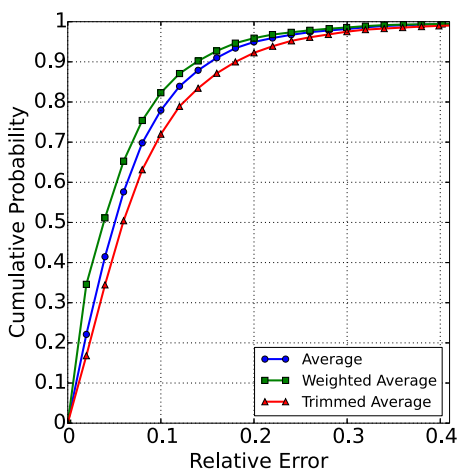


Fig. 6. Comparison between different prediction methods with the surrogate number q equal to 5.

8. Boston University Representative Internet Topology Generator, http://www.cs.bu.edu/brite/index.html

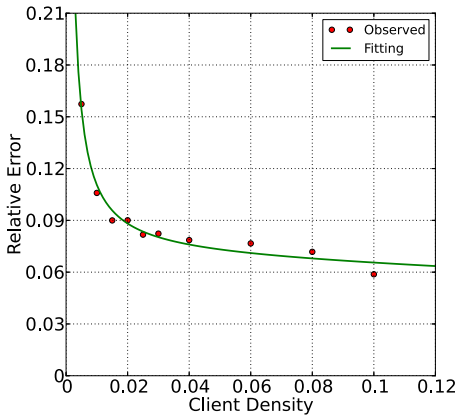


Fig. 7. The observed data and the fitting curve for the average relative errors under different client densities.

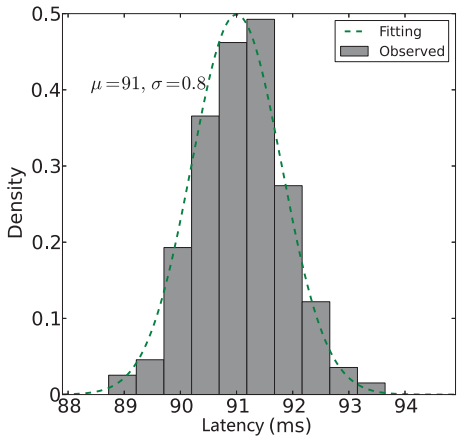


Fig. 9. The distribution of the observed means for the 400 resampling sequences and the fitting distribution when the sample ratio is equal to 0.15.

when using the average, the weighted average, and the trimmed average, respectively. As shown in Fig. 6, the weighted average achieves the highest accuracy among the three methods, followed by the average and the trimmed average worst. In the following experiments, we only conduct the prediction based on the weighted average method.

Effects of the Clients' Density on Prediction Accuracy. Intuitively, the more the clients, the higher *SSL*'s prediction accuracy. To explore the effects of the clients' density on the prediction accuracy, we changed the number of the clients and computed the average relative error accordingly.

Fig. 7 shows the observed data and fitting curve for the average relative errors under different client densities. In the figure, the red dots are the observed average relative errors under specific client densities, and the green solid curve is the fitting curve based on them. The results clearly demonstrate that the average relative error reduces as the density of the clients increases, following the negative exponential distribution and the convergence point can achieve when the client density is about 2 percent. When 2 percent of the hosts have installed specific softwares and act as the clients, the average relative error can be 9 percent. For large companies, e.g., Google, it is possible for them to embed plugins in web browsers etc. and get much higher client density.

Inferring the Statistical Information between Two Host Sets. To evaluate *SSL*'s performance in inferring the statistical information between two host sets, we randomly sample the nodes in the topology and obtain two host sets. Each set contains 100 hosts, that is, the universal set include 10,000 edges to be measured. The actual statistical latency between the two sets is about 91 ms.

In the simulation, we set the client density to 10 percent, that is, there exist 500 clients among the overall nodes. For the clients, their capacity is positively correlated to the overall bandwidth of their links. To reduce the measurement overhead, *SSL* first samples the 10,000 edges, assigns the sampled tasks to the clients, and uses the weighted mean of the sampled edges to represent the mean of the universal set. During the resampling process, we repeat the resampling process for $B = 400$ times.

Fig. 8 shows the resampling means and its 95 percent confidence intervals with the sampled ratio, that is, $|P|/mn$. From Fig. 8, we can conclude that the more edges *SSL* measures, the narrower the range of 95 percent confidence intervals, and the higher accuracy of the inferred statistical information. Moreover, the results of different measurements are within 2.7 percent with each other, that is, *SSL* is precise.

Without loss of generality, we extract the case when the sample ratio is equal to 0.15. For the means of the 400 resampling sequences, their distribution is shown in Fig. 9 where the histogram is the distribution of observed means and the green dash curve is the fitting curve for the histogram, which satisfies a normal distribution with the mean value as 91 ms and the standard deviation as 0.8 ms. The mean value is exactly the statistical latency between the two sets. Based on Fig. 9, we can draw the conclusion that the statistical latency between the two sets is 91 ms with the 95 percent confidence interval as $[89.4, 92.6]$ ms, which is shown in Fig. 8.

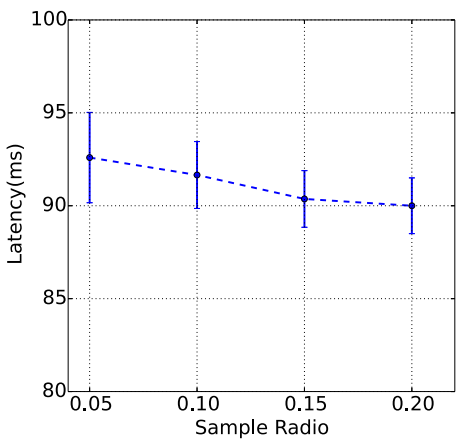


Fig. 8. Comparison of the resampling means and 95 percent confidence intervals under different sampling ratios.

6 Related Work

There has been considerable amount of work on network distance estimation [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [29]. However, none of them satisfies the requirements of statistical latency measurement. We summarize these methods into three categories: coordinate-based methods, infrastructure-based methods, and path-fitting methods.

Ng et al. [11] modeled the Internet as a geometric space and predict the latency using the distance in coordinate space. Dabek et al. [13] proposed a distributed network coordinate system. However, these coordinate-based methods are not always accurate due to the existence of asymmetric routing and Triangle Inequality Violation (TIV) in the Internet. Apart from these coordinate-based methods, Francis et al. [10] deployed Tracers across the Internet and estimated the distance between any two hosts using the sum of the distance from each host to its nearest Tracer, and the distance between the two Tracers. Madhyastha et al. [21] predicted end-to-end path performance by composing the performance of measured segments of the Internet paths. However, these path-fitting methods always rely on numerous deployed servers in the Internet, incurring high extra costs. Furthermore, it needs massive amount of active probing and thus it is not light-weight. On the other hand, infrastructure-based methods always require the support of infrastructure in the Internet [12], [18]. Gummadi et al. [12] estimated the latency between two end hosts by measuring the latency between their authoritative name servers. However, it requires DNS in the Internet to be open and recursive, which brings about security issues [22], [23].

There are other related works about measuring the Internet from the edge. Sanchez et al. [30] designed the Dasu platform, and now the platform is mainly used to measure user behavior. Shavitt et al. [31] proposed DIMES to measure the Internet topology. Different from these works, our paper focuses on end-to-end latency measurement.

7 Conclusion

The existing methods for latency estimation mainly focus on the realtime value. However, the statistical latency measurement, which is defined as the average latency between two groups of hosts across a time period, is also of great significance to many applications, such as service recommendation, site provisioning and SLA compliance monitoring. This paper presents a novel statistical latency measurement method called *SSL* for large scale networks. *SSL* mainly takes advantage of widely distributed but volatile clients to measure the Internet, including the method to predict the latency between two arbitrary hosts, the way to reduce the measurement overhead at scale and the mechanism to satisfy the limited source constraints. *SSL* neither incurs extra deployment cost nor brings out security problems. Moreover, we have also explored *SSL*'s choice design and found that when measuring the latency between two hosts, the prediction accuracy is much higher when adopting the weighted mean of the latencies from the closest 5 clients of one host to the other host.

Future works include developing software plugins to support *SSL*, and extending *SSL* to measure other network characteristics, such as the packet loss and the bandwidth.

Acknowledgments

This work was supported in part by the National Key Research and Development Program under Grant no. 2016YFB1000102, in part by the National Natural Science Foundation of China under Grant no. 61672318, 61631013, 61402343, 61529101, 61371166, 61571215, 61671235, 61902178, in part by the EU FP7 QUICK project under Grant Agreement No. PIRSES-GA-2013-612652, in part by the Natural Science Foundation of Jiangsu under Grant BK20190295, and in part by the projects of Tsinghua National Laboratory for Information Science and Technology (TNList).

References

[1] X. Wang, J. Zhu, and Y. Shen, "Network-aware QoS prediction for service composition using geolocation," *IEEE Trans. Services Comput.*, vol. 8, no. 4, pp. 630–643, Jul. 2015.

[2] V. Shah and G. de Veciana, "High-performance centralized content delivery infrastructure: Models and asymptotics," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1674–1687, Oct. 2015.

[3] W. Du, Y. Liao, N. Tao, P. Geurts, X. Fu, and G. Leduc, "Rating network paths for locality-aware overlay construction and routing," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1661–1673, Oct. 2015.

[4] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 371–382.

[5] J. C. Mogul and R. R. Kompella, "Inferring the network latency requirements of cloud tenants," in *Proc. 15th USENIX Conf. Hot Topics Operating Syst.*, 2015, pp. 24–24.

[6] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Multiobjective monitoring for SLA compliance," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 652–665, Apr. 2010.

[7] H. Yin, X. Zhang, T. Zhan, Y. Zhang, G. Min, and D. O. Wu, "NetClust: A framework for scalable and pareto-optimal media server placement," *IEEE Trans. Multimedia*, vol. 15, no. 8, pp. 2114–2124, Dec. 2013.

[8] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware Web service recommendation by collaborative filtering," *IEEE Trans. Services Comput.*, vol. 4, no. 2, pp. 140–152, Apr. 2011.

[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2015, pp. 1346–1354.

[10] P. Francis, et al., "IDMaps: A global internet host distance estimation service," *IEEE/ACM Trans. Netw.*, vol. 9, no. 5, pp. 525–540, Oct. 2001.

[11] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2002, pp. 170–179.

[12] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proc. 2nd ACM SIGCOMM Workshop Internet Meas.*, 2002, pp. 5–18.

[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2004, pp. 15–26.

[14] M. Costa, M. Castro, A. Rowstron, and P. Key, "PIC: Practical internet coordinates for distance estimation," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 178–187.

[15] Y. Mao and L. K. Saul, "Modeling distances in large-scale networks by matrix factorization," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 278–287.

[16] Y. Mao, L. K. Saul, and J. M. Smith, "IDES: An internet distance estimation service for large networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2273–2284, Dec. 2006.

[17] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *Proc. 4th USENIX Conf. Netw. Syst. Des. Implementation*, 2007, pp. 22–22.

[18] D. Leonard and D. Loguinov, "Turbo king: Framework for large-scale internet delay measurements," in *Proc. 27th IEEE Conf. Comput. Commun.*, Apr. 2008, pp. 31–35.

[19] A. Jain and J. Pasquale, "Internet distance prediction using node-pair geography," in *Proc. 11th IEEE Int. Symp. Netw. Comput. Appl.*, Aug. 2012, pp. 71–78.

[20] Y. Chen, et al., "Phoenix: A weight-based network coordinate system using matrix factorization," *IEEE Trans. Netw. Service Manage.*, vol. 8, no. 4, pp. 334–347, Dec. 2011.

[21] H. V. Madhyastha, et al., "iPlane: An information plane for distributed services," in *Proc. 7th Symp. Operating Syst. Des. Implementation*, 2006, pp. 367–380.

[22] To manually test an IP address. (2017). [Online]. Available: http://www.openresolver.com/

[23] The danger of open recursive DNS resolvers. (2013). [Online]. Available: http://www.techmaish.com/the-danger-of-open-recursive-dns-resolvers/
[24] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2716–2720.
[25] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and evaluating large-scale CDNs paper withdrawn at Mirosoft's request," in *Proc. 8th ACM SIGCOMM Conf. Internet Meas.*, 2008, pp. 15–29.
[26] B. Efron, "Bootstrap methods: Another look at the Jackknife," *Ann. Statist.*, vol. 7, no. 1, pp. 1–26, 1979.
[27] Introduction to the Bootstrap. (2016). [Online]. Available: http://galton.uchicago.edu/∼eichler/stat24600/Handouts/bootstrap.pdf
[28] J. B. Orlin, "Max flows in O(NM) time, or better," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, 2013, pp. 765–774.
[29] H. Huang, et al., "Network distance prediction for enabling service-oriented applications over large-scale networks," *IEEE Commun. Mag.*, vol. 53, no. 8, pp. 166–174, Aug. 2015.
[30] M. A. Sanchez, et al., "A measurement experimentation platform at the Internet's edge," *IEEE/ACM Trans. Netw.*, vol. 23, no. 6, pp. 1944–1958, Dec. 2015.
[31] Y. Shavitt and E. Shir, "DIMES: Let the internet measure itself," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 71–74, Oct. 2005.

Xu Zhang is an Associate Researcher in the School of Electronic Science and Engineering, Nanjing University, Nanjing, Jiangsu, China. He received the BSc degree in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012 and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2017. He was a visiting student in the Department of Electrical and Computer Engineering, University of Florida, Florida, from 2015 to 2016. His research interests include content delivery networks, network measurement, and cloud computing.

Hao Yin received the BS, ME, and PhD degrees from the Huazhong University of Science and Technology, Wuhan, China, in 1996, 1999, and 2002, respectively, all in electrical engineering. He is a professor in the Research Institute of Information Technology (RIIT), Tsinghua University. He was elected as the New Century Excellent Talent of the Chinese Ministry of Education in 2009, and won the Chinese National Science Foundation for Excellent Young Scholars in 2012. His research interests span broad aspects of multimedia communication and computer networks.

Dapeng Oliver Wu (S'98-M'04-SM'06-F'13) received the BE degree in electrical engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1990, the ME degree in electrical engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 1997, and the PhD degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 2003. He is a professor in the Department of Electrical and Computer Engineering, University of Florida, Florida. His research interests include the areas of networking, communications, signal processing, computer vision, machine learning, smart grid, and information and network security. He is a fellow of the IEEE.

Haojun Huang received the BSc degree from the School of Computer Science and Technology, Wuhan University of Technology, in 2005 and the PhD degree from the School of Communication and Information Engineering, University of Electronic Science and Technology of China, in 2012. He is a lecturer within the College of Electronic Information, Wuhan University, China. He was a postdoctoral researcher in the Research Institute of Information Technology (RIIT), Tsinghua University from 2013 to 2015. His research interests focus on big data, mobile Internet, wireless communication, and ad hoc networks.

Geyong Min received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995 and the PhD degree in computing science from the University of Glasgow, United Kingdom, in 2003. He is a professor of high performance computing and networking in the Department of Mathematics and Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, United Kingdom. His research interests include future internet, computer networks, wireless communications, multimedia systems, information security, high performance computing, ubiquitous computing, modelling, and performance engineering.

Ying Zhang received the PhD degree from the EECS Department, University of Michigan. She is a senior research scientist in the Networking and Mobility Lab, Hewlett Packard Labs since 2014. Prior to that, she is a senior researcher in Ericsson Research Silicon Valley Lab from 2009. Her research interests is networking and systems, including network function virtualization, software defined networking, cloud and data centers, internet routing and measurement, and network security.