Schedule Distributed Virtual Machines in a Service Oriented Environment

Lizhe Wang†, Gregor von Laszewski†, Marcel Kunze‡, Jie Tao‡
† Pervasive Technology Institute, Indiana University Bloomington, IN 47404
‡ Steinbuch Center for Computing, Karlsruhe Institute of Technology, 76344 Eggenstein-Leopoldshafen, Germany

***Abstract*—Virtual machines offer unique advantages to the scientific computing community, such as Quality of Service(QoS) guarantee, performance isolation, easy resource management, and the on-demand deployment of computing environments. Using virtual machines as a computing resource within a distributed environment, such as Service Oriented Architecture (SOA), creates a variety of new issues and challenges that must be overcome. Traditionally, parallel task scheduling algorithms only focus on handling CPU resources. Using of a virtual machine, however, requires the monitoring and management of additional resource properties. Additionally, CPU, memory, storage, and software licenses must also be considered within the scheduling algorithm. The objective of this paper is to address these challenges of a multi-dimensional scheduling algorithm for virtual machines within a SOA. To do this, we deploy a testbed SOA environment composed of virtual machines which are capable of being registered, indexed, allocated, accessed, and controlled by our new parallel task scheduling algorithm.**

***Index Terms*—Service oriented architecture, Virtual machine, Scheduling algorithm**

I. INTRODUCTION

Distributed computing environments and algorithms have been developing with impressive progress for many years. Despite these great advances in distributed computing, several challenges still remain in actual distributed infrastructure implementation, for example, Quality of Service (QoS) support, customized computing environment provision, and flexible IT infrastructure deployment.

Virtual machine technology recently has emerged as a hot topic offering a variety of excellent advantages over tradtional computing, such as performance isolation and customized computing environment provision. Employing virtual machines as computing resources in a distributed system, therefore, is an interesting solution for the problems discussed above. Service Oriented Architecture (SOA) is a distributed computing environment where components are packaged as independent services. These services interoperate and communicate with each other using standardized protocols. Currently, SOA is one of the most popular building frameworks for distributed systems.

This paper is devoted to the topic of scheduling virtual machines in a SOA environment, which brings new research issues, for example,

- resource provision is more complex, since virtual machines and their host resources both need to be considered
- compared with resource allocation in the traditional parallel & distributed system, virtual machines contain more properties which are used for scheduling, for example, memory size, software packages, and access to specific devices

In this paper, models for parallel tasks and virtual machine based SOA environment are studied. A new parallel scheduling algorithm, the Multi-Dimensional Scheduling Algorithm (M-DSA) is designed and implemented in a SOA environment. Simulation results show M-DSA's performance improvement when compared to the Random Resource Allocation Algorithm (RRAA). A test case, bio-sequence alignment application, runs in a real SOA environment scheduled with M-DSA.

The paper is organized as follows: Section II presents the background of the related research – virtual machine technology, the SOA environment and the task scheduling algorithms. Section III formally defines the research of scheduling virtual machines in a SOA environment: the target system model, the parallel task model and the problem specification. Section IV presents the M-DSA. Section V gives a performance evaluation on the M-DSA simulation in a SOA environment. Section VI investigates building a SOA environment on distributed virtual machines and the implementation of M-DSA in a resource broker. Section VII concludes the paper and points out future work.

II. RELATED WORK

A. Virtual Machine

A **Virtual Machine** (VM) is a software artifact that executes other software in the same manner as the machine for which the software is developed and executed [12]. The software that supports multiple virtual machines on the same resource is termed as a **Virtual Machine Monitor** (VMM) or hypervisor. The server that provides resources for multiple virtual machines is termed as a **host resource**.

Test results show that employing virtual machines does not induce a significant performance reduction – applications lose around 5%-15% of the performance when compared to the same cases on real machines [18]. For that reason employing virtual machine as a computing resource can deliver various advantages, for example, on-demand creation and customization, QoS guarantee and performance isolation, legacy software support, and easy resource management.

Popular virtual machine examples are Xen [2] and VMware [15] products. Currently, virtual machines are widely employed to build IT-infrastructure, distributed computing platforms, and parallel systems. Examples exist in the Globus

Virtual Workspace and Nimbus [4], workflow systems on distributed virtual machines [17], the In-Vigo project [1], the Virtuoso project [11], Virtual resource marketing [3], the OpenNebula [14] and the Cumulus project [19].

B. *Web Service and SOA*

Web services are application components using open protocols to communicate, for example, the Simple Object Access Protocol (SOAP) standard. Web services are self-contained and self-describing – Web services are described in the Web Services Description Language (WSDL)and can be discovered using the Universal Description, Discovery and Integration (UDDI).

Web services can be used to implement a Service Oriented Architecture (SOA). In general, a SOA [8] contains a service provider, a service broker, and a service requester. It is believed that the SOA can help businesses respond more quickly and cost-effectively to changing market conditions. The SOA promotes the objective of separating users from the service implementations. Services can therefore be executed on various distributed platforms and be accessed across networks making them ubiquitous to the end user. This can also maximize the reuse of services and enable the interoperability of distributed components and entities.

C. *Task Scheduling Algorithm in Parallel & Distributed Systems*

The problem of task scheduling is the distribution of the tasks to Processing Elements (PEs) to achieve some performance goals, e.g., minimizing execution time, minimizing communication delays, or maximizing resource utilization. Task scheduling algorithms are typically classified into two subcategories: static task scheduling algorithms and dynamic task scheduling algorithms.

In static task scheduling algorithms, the assignment of tasks to resources is performed before applications are executed. Information about task execution time and communication delay is assumed to be known at compilation time. Since task scheduling is an NP-complete problem, a large number of heuristic algorithms [21], [16] have been developed. Heuristic algorithms mainly rely on rules-of-thumb to guide the scheduling process to reach near optimal solutions.

Dynamic task scheduling algorithms are based on the redistribution of tasks among PEs during execution time. In order to improve the performance of applications, the distribution is performed by transferring tasks from the heavily loaded PEs to lightly loaded PEs [5], [9], [20].

The advantage of dynamic load balancing algorithms over static scheduling algorithms is that the system does not require the pre-knowledge for task allocation. However, dynamic task scheduling algorithms may cause some run-time overhead.

Many static task scheduling algorithms are based on the list scheduling algorithm [10], [7], [6]. List based scheduling algorithms assign priorities to tasks and sort tasks into a list ordered in decreasing priority. Then tasks are scheduled based on the priorities.

III. Scheduling distributed virtual machines in a distributed system: problem definition

A. *What's new?*

The studied system contains a number of computing servers (or host resources), each of which supplies several virtual machines. Computing resources of a server, e.g., CPU and memory, are divided between the host's virtual machines. Each virtual machine is pre-installed with some application level software packages. Tasks will be allocated to some virtual machine that can provide the proper resources (CPU and memory) and software.

The aforementioned distributed computing environment is more complex than traditional distributed systems in that:

- Computing resources required to be scheduled are multiple dimension, for example, CPU bandwidth, memory, and software licences. In traditional scheduling environment, only processor resources are considered for resource allocation.
- Resource allocation should be considered with more restrictions, for example, some applications can only be scheduled to certain virtual machines that provide the required application execution environment. This scenario does not exist in a traditional distributed system.

This section discusses the formal specification of the research issue – scheduling parallel tasks for virtual machine allocation in a SOA.

B. *Target System Model*

The target system, where resources are managed in a SOA environment, can be described as: $G = \{H, V, L\}$, where:

- Host resource: $H = \{h_i\}$, $1 \leq i \leq R$
  H is the set of R hosts in the target system. R is the total amount of the hosts. Each host h_i has 2 properties:
  - $h_i.CPUPerformance$ is the value of CPU bandwidth of the host,
  - $h_i.MemorySize$ is the value of memory size of the host.
- Virtual machines: $V = \{v_i\}$, $1 \leq i \leq M$
  V is the set of M virtual machines. M is the total amount of the virtual machines. Each virtual machine v_i has 1 property:
  - $v_i.Software$ is the software installed on the virtual machine, this can be a set of software.
- Virtual machine affiliation $L = [L_{ik}]$
  L is a $R \times M$ matrix. L_{ik} represents the host H_i has L_{ik} virtual machine v_k, $1 \leq i \leq R$ and $1 \leq k \leq M$.

$$L_{ik} = \begin{cases} 0 & \text{if host } h_i \text{ has not the virtual machine } v_k \\ 1 & \text{if host } h_i \text{ has the virtual machine } v_k \end{cases}$$

C. *Parallel Task Model*

Parallel tasks are modeled as $T = \{J, <\}$, where,

- $J = \{j_i\}$, , $1 \leq i \leq N$
  J is the set of N tasks to be scheduled. N is the total number of tasks in J. Each task j_i has 4 properties:

  - $j_i.Req\_Software$: software required, this can be a set of software,
  - $j_i.Req\_CPU$: CPU bandwidth required,
  - $j_i.Req\_Mem$: memory size required, and
  - $j_i.Req\_Time$: execution time required.
- $<$ is a partial order defined on J
  $<$ specifies operational precedence constraints. $j_i < j_k$ means that j_i must be completed before j_k can begin, $1 \leq i \leq N$, $1 \leq k \leq N$, $j_k, j_i \in J$.

D. Problem Definition

For parallel tasks $T = \{J, <\}$, the resource allocation array is defined as:

$$S = \{S_i\} = \{< h_{ij}, v_{ik}, t_{iq} >\}$$
$$1 \leq ij \leq R, 1 \leq ik \leq M, 1 \leq iq, h_{ij} \in H, v_{ik} \in V, 0 \leq t_{iq}$$

where,

- h_{ij}, v_{ik} are the host and the virtual machine allocated for the task j_i, and
- t_{iq} is the starting time of the task j_i

It should be noted that ij here is a single variable to represent an integer, it is not two dimensional subscript. ik and iq are the same case.

For the sake of easily specifying the problem the following definitions are introduced:

- TST: Task Starting Time
  Each task j_i, $1 \leq i \leq N$, has its own Job Starting Time: JST_i. Task Starting Time of parallel task T, TST, is defined as follows:
  $$TST = min\{JST_1, JST_2, ..., JST_N\}$$
- TFT: Task Finish Time
  Each task j_i , $1 \leq i \leq N$, has its own Job Finish Time: JFT_i. Task Finish Time of parallel task T, TFT, is defined as follows:
  $$TST = max\{JFT_1, JFT_2, ..., JFT_N\}$$
- TET: Task Execution Time
  Task Execution Time of parallel task T, TET, is defined as follows:
  $$TET = TFT - TST$$

The schedule of the parallel task $T = \{J, <\}$ on the target system $G = \{H, V, L\}$ is a function f which maps the parallel task J to the allocation array S:

$$f : J \rightarrow S, f \in F, F \text{ is the set of all feasible mappings.}$$

Therefore, the problem of parallel task scheduling on the target system can be defined as follows:
Given parallel tasks $T = \{J, <\}$ and the target system $G = \{H, V, L\}$, find a schedule f_{min}, $f_{min} \in F$, which gives the parallel task the minimum Task Execution Time TET.

IV. Multi-Dimension Scheduling Algorithm

A. Why is static scheduling algorithm used?

This section proposes a Multi-Dimension Scheduling Algorithm (M-DSA) for parallel task resource allocation, which belongs to the static scheduling algorithms classification. Static scheduling algorithms are adopted for distributed virtual machine allocation with the following considerations.

Firstly, the distributed virtual machines are prepared and required via resource reservations or mutual negotiation in the SOA. Therefore, the tasks' resource requirements can be gathered at either compile time or preparation time. This is the typical context for static scheduling algorithms.

Dynamic scheduling algorithms normally require the running task's load information. In the distributed virtual machine environment, virtual machines reserve and guarantee the resource allocation of the running tasks. Therefore, the load information is meaningless for scheduling algorithms.

Furthermore, dynamic scheduling algorithms sometimes demand task migration and the transfer of a running virtual machine over wide-area networks is expensive.

B. How does the M-DSA Differ from Existing List Based Scheduling Algorithms?

Task's requirements on host resources are multi-dimensional. In the parallel task model, hence the term "Multi-Dimension", the following are used – required CPU, required remory, required task execution time, and required software environment. In traditional list scheduling algorithms resources are modeled as PEs, which only count for the CPU bandwidth or the processor number. When multiple resource dimensions are considered, coordinated scheduling policies should be developed to adopt the multiple restrictions. The M-DSA is designed and implemented to handle the new requirements for task scheduling.

The M-DSA is a list based scheduling algorithm. Therefore, the primary goal of the algorithm is to schedule a task as early as possible when all the resources required by the job are available.

The M-DSA contains two steps:

- Task sorting
  Initially all the tasks of the parallel task should be sorted according to the dependencies between tasks.
- Resource allocation
  Then the sorted tasks are scheduled onto the target system. The proper host resources and virtual machines will be allocated to each task.

C. Task Sorting Algorithm

The parallel tasks $T = \{J, <\}$ can be represented by a Directed Acyclic Graph (DAG). Algorithm 1 shows the algorithm of building a DAG from parallel task $T = (J, <)$.

In Algorithm 1, the set of all the direct predecessors of each vertex is set to empty firstly. For each dependence $j_i < j_k$ of the set $<$, a directed arc from vertex j_i to j_k is created and the vertex j_i is added to the set of the direct predecessors of vertex j_k. Finally the created DAG is output.

Algorithm 2 presents the sorting algorithm for a DAG. In Algorithm 2, $IncomingDegree$ is the number of the incoming degree of a vertex. $Sequence\_Stack\_Vertex$ is a sequence stack for storing vertexes whose incoming degrees are 0. $Directpredecessors$ is the set of all the direct predecessors

Algorithm 1 Algorithm of DAG generation

```
BEGIN
FOR each vertex $j_i \in J$ DO
    $j_i.Directpredecessors \leftarrow \emptyset$
ENDFOR
FOR each dependence $j_i < j_k \in <$ DO
    create a directed arc from vertex $j_i$ to $j_k$
    Add $j_i$ to $j_k.Directpredecessors$
ENDFOR
output DAG
END
```

Algorithm 2 Task Sorting Algorithm

```
$v_i.IncomingDegree$: the number of the incoming degree of vertex $v_i$;
$Sequence\_Stack\_Vertex$: a stack that stores vertex;
$v_i.Directpredecessors$: the set of all the direct predecessors of the vertex $v_i$.
$v_i.Direcsuccessors$: the set of all the direct successors of the vertex
$v_i.Sorted\_Jobs\_List$: the list of sorted vertexes;
BEGIN
$Sorted\_Jobs\_List \leftarrow \emptyset$;
FOR each vertex $v_i \in$ DAG DO
    IF ($v_i.IncomingDegree = 0$)
        Add $v_i$ into the $stackSequence\_Stack\_Vertex$
    ENDIF
ENDFOR
WHILE ($Sequence\_Stack\_Vertex$ is not empty) DO
    $v \leftarrow$ pop $Sequence\_Stack\_Vertex$
    Add the vertex$v$ to the end of $Sorted\_Jobs\_List$
    remove $v$ and its arcs from DAG
    FOR each $v_j \in v.Direcsuccessors$ DO
        $v_j.IncomingDegree \leftarrow v_j.IncomingDegree - 1$
        IF ($v_j.IncomingDegree = 0$)
            Add $v_j$ into the stack$Sequence\_Stack\_Vertex$;
        ENDIF
    ENDFOR
ENDWHILE
IF (number of output vertex $< n$)
    ERROR ("Jobs dependencies Error. There is ring in the DAG.")
ELSE
    output $Sorted\_Jobs\_List$
ENDIF
END
```

of a vertex. $Direcsuccessors$ is the set of all the direct successors of a vertex. $Sorted\_Jobs\_List$ is the list of sorted vertexes.

First the algorithm initializes the list $Sorted\_Jobs\_List$ to empty and push all the vertexes whose incoming degrees are 0 into the sequence stack $Sequence\_Stack\_Vertex$. While the $Sequence\_Stack\_Vertex$ is not empty, the algorithm does the loop as follows: it pops a vertex v out of the stack $Sequence\_Stack\_Vertex$ and adds the vertex v to the end of the list $Sorted\_Jobs\_List$, removes the vertex v and its arcs from the DAG; for each vertex v_j in the set $Direcsuccessors$ of the vertex v, it reduces the incoming degree of the vertex v_j with 1; at this moment, if the incoming degree of v_j is 0, pushs the vertex v_j into the stack $Sequence\_Stack\_Vertex$ and then repeats the loop until there is no vertex in the $Sequence\_Stack\_Vertex$ anymore.

Finally, if the number of the output vertexes is the same as the number of the vertexes in the DAG, output the list $Sorted\_Jobs\_List$, it means that the vertexes of the DAG are sorted successfully, otherwise the input data DAG is incorrect.

D. Resource Allocation Algorithm

In Algorithm 3, $Sorted\_Jobs\_List$ is the sorted job list which is the result of Algorithm 3. $j_i.Directpredecessors$ is the set of all the direct predecessors of the task j_i, this is the result of Algorithm 1. T_0 is the possibly earliest task starting time of a task. TFT is the Task Finish Time of a task. h is the allocated host for a job. vm is the allocated virtual machine for a task in the loop. TST is the Task Starting Time for the task execution. TET is the total Task Execution Time of all the tasks.

For a certain task selected from $Sorted\_Jobs\_List$, the algorithm searches all the hosts for suitable a virtual machine resource. If it finds one for the first time, the information of the suitable resource is stored in the variables h, vm, and TST. If it finds another suitable resource with information of h_x, vm_x, and a earlier task staring time TST_x, the h_x, vm_x and TST_x would be selected as resource allocation for task j_i.

Here, we define the **suitable resource** h and vm for task j as follows:

$$j.Req\_CPU \leq h.CPUPerformance$$
$$j.Req\_Mem \leq h.MemorySize$$
$$j.Req\_Software \subseteq vm.Software$$

The **more suitable resource** means resource allocation with consideration of load balance. In other words, a host with light computing load profile is more suitable than a host with heavy computing load profile. We define a parameter, RLP_t – Resource Load Profile, to represent the load status of the target system at certain time, t.

$$RLP(t) = \frac{\sum_j (T_j.Req\_CPU \times T_j.Req\_Mem)}{\sum_i (h_i.CPUPerformance \times h_i.MemorySize)}$$

Where,
$h_i.CPUPerformance$ is the value of CPU bandwidth of the host h_i,
$h_i.MemoerySize$ is the value of memory size of the host h_i,
T_j is the task which occupies one host resource at time t,
$T_j.Req\_CPU$ is the value of required CPU of task T_j, and
$T_j.Req\_Mem$ is the value of required memory of task T_j.

$RLP(t)$ shows, at certain time t, how much resources are occupied. In other words, it reflects, how busy the system is in certain time transaction.

V. Simulation and Performance Evaluation

In this section, the performance of parallel task scheduling algorithm – D-MSA is analyzed by simulation. TETs of

Algorithm 3 Resource Allocation Algorithm

```
$Sorted\_Jobs\_List$: the list of sorted tasks
$j_i.Directpredecessors$: the set of all the direct predecessors of the task $j_i$.
$T_0$: the possibly earliest job starting time
$TFT$: Task Finish Time of a task
$h$: allocated host for the task
$vm$: allocated virtual machine for the task
$TST$: Task Starting Time for the task execution
$TET$: the total Task Execution Time of all the jobs
BEGIN
FOR each task $j_i$ in $Sorted\_Jobs\_List$ DO
    $found\_FreeResource$ = False
    $TST = 0$
    IF ($j_i.DirectPredecessors = \emptyset$)
        $T_0 = 0$
    ELSE
        $T_0$ = max {$TFT$ of $j_i.DirectPredecessors$}
    ENDIF
    FOR each host DO
        IF (find a suitable host $h_x$ with $vm_x$ ($L_{h,vm} = 1$), and task starting time $TST_x$ after $T_0$)
            IF ($found\_FreeResource$ = False)
                $h = h_x$
                $vm = vm_x$
                $TST = TST_x$
                $found\_FreeResource = $ True
            ELSE
                IF ($TST = TST_x$)
                    IF ($h_x$ is more suitable than $h$)
                        $h = h_x$
                        $vm = vm_x$
                    ENDIF
                ELSE IF ($TST > TST_x$)
                    $h = h_x$
                    $vm = vm_x$
                    $TST = TST_x$
                ENDIF
            ENDIF
        ENDIF
    ENDFOR
    schedule $< h, vm, TST >$ to task $j_i$
    Update the host $h$ resource allocation information with $< vm, TST, j_i >$
ENDFOR
$TFT = max\{TFT$ of all the tasks$\}$
output $TFT$
END
```

different sizes of parallel tasks are examined with different load profiles of the target system.

A. Simulation Environment

The simulation environment is composed of two modules:

- Task Module: Task Module generates the parallel tasks to be scheduled. The Task Module includes two components:
  - DAG Generator: DAG Generator generates DAG of parallel tasks, including dependencies between tasks. Users can specify parameters for task generation, e.g., task number. In this simulation, 2 sets of parallel tasks are generated whose task number are 100 and 200 respectively.
  - Task Requirement Generator: Task Requirement Generator randomly generates resource requirements for tasks.
- Resource Module: The Resource Module generates the target system for simulation. There are two components in the Resource Module: VM Generator and Host Generator. VM Generator generates the virtual machines and their installed software. Host Generator generates the hosts with resource properties.

B. Simulation results and Evaluation

The M-DSA is tested in the simulation environment discussed above. A performance comparison between the M-DSA and the RRAA is performed to justify the performance improvement of the M-DSA. The Random Resource Allocation Algorithm (RRAA) sorts the parallel tasks input and randomly selects hosts and virtual machines for parallel tasks, which can fulfill the task resource requirements. The RRAA is simulated and tested in the same environment.

A complete performance comparison is done in various simulation scenarios:

- different RLPs: 0%, 5%, 10%[1] and
- different task number: 100 and 200.

The host number is fixed at 100. The simulation results are shown in Figure 1. It can be concluded that the M-DSA can gain at least 20% performance improvement compared with the RRAA.

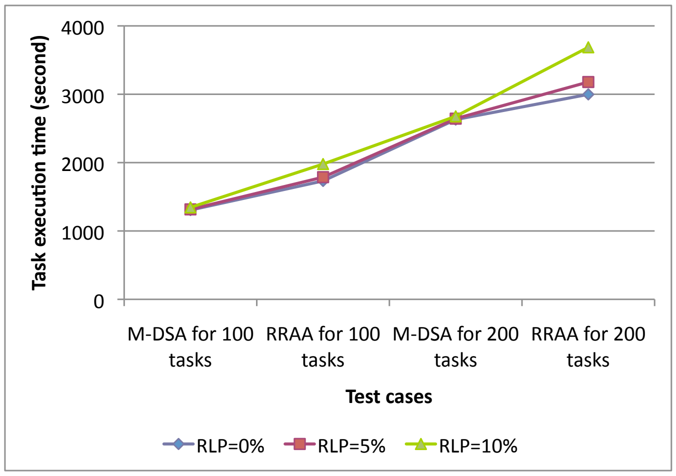

Fig. 1. Performance comparison of the M-DSA and the RRAA

[1]The scenario of $RLP = 0$ means that no tasks run on distributed resources. In this simulation, it is generated to evaluate the performance of scheduling algorithms in some lightly loaded use cases.

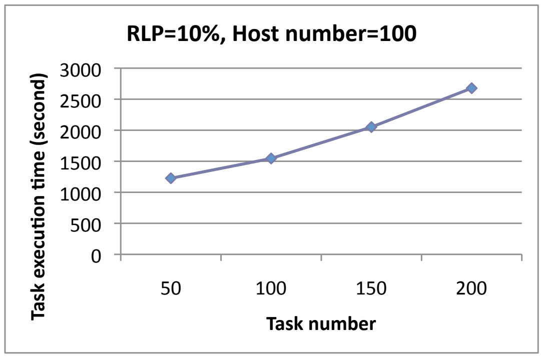

Fig. 2. Performance of the M-DSA when the task number increases

The scalability of the M-DSA is also studied in the simulation. The host number is fixed to 100, Figure 2 shows the performance of the M-DSA when the task number increases. The Task Execution Time increased near-linearly with the increasing of task number. The task number is fixed to 200, Figure 3 shows the performance of the M-DSA when the host number increases. The Task Execution Time also increased near-linearly with the increasing of host number.

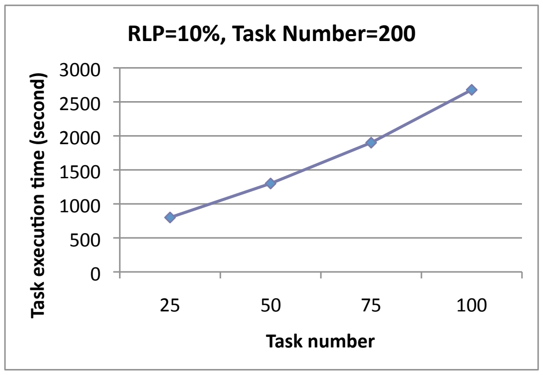

Fig. 3. Performance of the M-DSA when the host number increases

VI. Implementation of M-DSA in a SOA

A. *SOA Computing Environment with Distributed Virtual Machine Resources*

1) Overview: The SOA computing environment with distributed virtual machines contains a UDDI server, distributed host resources and users. Virtual machines are supported by host resources and installed with various types of application software. The host resources are interfaced with a specific Web service that publishes resource information on the UDDI server. Users search resource information on the UDDI service and locate proper virtual machines for task execution.

2) Development Environment: The SOA environment was built with the Java Web Services Developer Pack (JWSDP) and the Sun Java System Application Server (SJSAS).

We use GridSAM [13] to provide a Web Service for submission and monitoring jobs that are managed by a variety of Distributed Resource Managers (DRM). GridSAM Web service can be embedded into software applications that requires job submission and monitoring capabilities.

Currently no public UDDI registry exists, therefore a private Registry Server should be configured. It is created with Sun Application Server 8.2 and Java WSDP 1.5. The UDDI service provides functionality to publish host resource information.

GridSAM is a Web service that can wrap application software packages. Each virtual machine is installed with GridSAM and users are allowed to access software packages for invocation.

In order to register and update the information of the host on the UDDI Registry Server, a client of Java API for XML Registry (JAXR) is implemented on the host. The JAXR client uses the JAXR API, which is a client program for accessing registries. In this work, the Host Resource Information includes the CPU performance, memory capacity, virtual machines with associated host and services (software) installed, and all the reserved resource units of the host.

B. *Resource Broker with M-DSA*

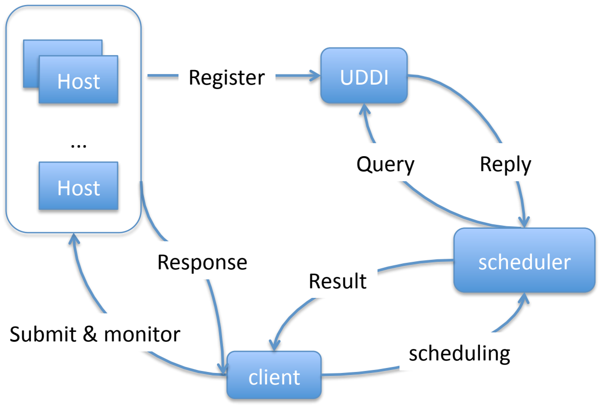

Fig. 4. Scheduler in the SOA with distributed virtual machines

1) Overview: Figure 4 shows the resource broker in the SOA environment. The Broker module operates as a server for all the end users and works as a client for the UDDI Registry Server. When a user submits a parallel task to the Broker, the Broker queries the UDDI Registry Server and obtains information about all the resources from the UDDI Registry Server in XML-based format. The Broker then allocates the resource for each job with the M-DSA Algorithm and returns the results to the user.

2) Implementation of Resource Broker: The Broker is a mediator between Client and UDDI Registry Server. Figure 5 shows the architecture of the Broker.

The Broker consists of 3 components: client interface, resource requester, and scheduler. The client Interface is responsible for the communications with clients. The resource requester queries the UDDI Registry Server to obtain the information about all the current suitable resources. The scheduler uses the M-DSA to allocate resources for parallel tasks.

3) Task Scheduling with Resource Broker: The task scheduling scenario is described in the following steps:

1 Each Host registers the necessary information on the UDDI Registry Server with the resource information, for example, host name, CPU performance, memory size,

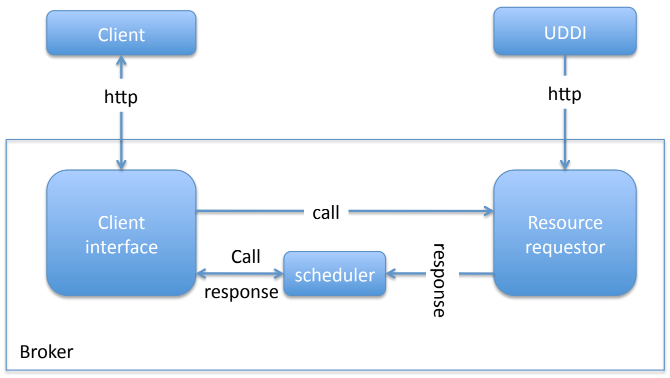

Fig. 5. Resource Broker

VMs of the host: VM Name and software offered by each VM (including URL information), and all the Occupied resource units on the host: Start time, End time, Occupied CPU & memory.

2 A user sends a parallel task in JDSL to the Broker. JSDL (Job Submission Description Language) is the standard of Global Grid Form (GGF).
3 The Broker queries the UDDI Registry Server for the information of required resources.
4 The UDDI Registry Server sends all the information of the required host resources to the Broker.
5 The Broker organizes the information of host resources and allocates the suitable resource for parallel tasks by using M-DSA.
6 The Broker returns scheduling result to the client.
7 The broker sends parallel tasks to scheduled resources.
8 Each host updates its Host Resource Information on the UDDI Registry Server.

VII. Conclusion and Future Work

It has been widely accepted that virtual machines can be employed as computing resources to build a distributed system for various reasons. SOA is one of the most popular architectures for building a distributed computing environment. This paper focuses on the research aspect of scheduling distributed virtual machines in a SOA environment.

We declare our contribution as follows:

- develop a M-DSA for task scheduling in a virtual machine based SOA environments
- make performance evaluation on the M-DSA by simulating various use scenarios
- build a SOA environment based on distributed virtual machines and implement M-DSA in a SOA environment,
- deliver a real use case to justify the M-DSA implementation in a SOA environment

We plan to continue developing M-DSA with data allocation considerations, and build a SOA environment on a heterogeneous networking, such as D-Grid project in the future.

Acknowledgment

Work conducted by Gregor von Laszewski and Lizhe Wang is supported (in part) by NSF CMMI 0540076 and NSF SDCI NMI 0721656.

References

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, Mi. Zhao, L. Zhu, and X. Zhu. From virtualized resources to virtual computing Grids: the In-VIGO system. *Future Generation Comp. Syst.*, 21(6):896–909, 2005.
[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177, New York, U. S. A., Oct. 2003.
[3] D. E. Irwin, J. S. Chase, L. E. Grit, A. R. Yumerefendi, D. Becker, and K. Yocum. Sharing networked resources with brokered leases. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 199–212, 2006.
[4] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: achieving quality of service and quality of life in the Grid. *Scientific Programming*, 13(4):265–275, 2005.
[5] Y. Lee and A. Y. Zomaya. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Trans. Computers*, 56(6):815–825, 2007.
[6] G. Liu, K. Poh, and M. Xie. Iterative list scheduling for heterogeneous computing. *J. Parallel Distrib. Comput.*, 65(5):654–665, 2005.
[7] A. Mtibaa, B. Ouni, and M. Abid. An efficient list scheduling algorithm for time placement problem. *Computers & Electrical Engineering*, 33(4):285–298, 2007.
[8] E. Newcomer and G. Lomow. *Understanding SOA with Web services*. Addison-Wesley, 2004.
[9] H. Park and B. Kim. Optimal task scheduling algorithm for cyclic synchronous tasks in general multiprocessor networks. *J. Parallel Distrib. Comput.*, 65(3):261–274, 2005.
[10] B. Shirazi, M. Wang, and G. Pathak. Analysis and evaluation of heuristic methods for static task scheduling. *J. Parallel Distrib. Comput.*, 10(3):222–232, 1990.
[11] A. Shoykhet, J. Lange, and P. Dinda. Virtuoso: a system for virtual machine marketplaces. Technical Report NWU-CS-04-39, Northwest University, July 2004.
[12] J. Smith and R. Nair. *Virtual machines: versatile platforms for systems and processes*. The Morgan Kaufmann, 2003.
[13] GridSAM: Grid Job Submission and Monitoring Web Service [URL]. http://gridsam.sourceforge.net/, access on Nov. 2007.
[14] OpenNEbula Project [URL]. http://www.opennebula.org/.
[15] VMware virtualization technology [URL]. http://www.vmware.com.
[16] Lizhe Wang, Wentong Cai, Bu-Sung Lee, Simon See, and Wei Jie. Resource Co-allocation for Parallel Tasks in Computational Grids. In *International Workshop on Challenges of Large Applications in Distributed Environments (CLADE 2003)*, pages 88–96, 2003.
[17] Lizhe Wang and Marcel Kunze. On the design of virtual environment based workflow system for grid computing. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops*, pages 212–218, Washington, DC, USA, 2006. IEEE Computer Society.
[18] Lizhe Wang, Marcel Kunze, and Jie Tao. Performance evaluation of virtual machine based Grid workflow system. *Concurrency and Computation: Practice and Experience*, 20(15):1759–1771, 2008.
[19] Lizhe Wang, Marcel Kunze, and Jie Tao. Scientific cloud computing: early definition and experience. In *Proceedings of the IEEE International Conference on High Performance Computing and Communications*, pages 795–803, Dalian, China., Sep. 2008.
[20] M. Wang, R. Kotagiri, and J. Chen. Trust-based robust scheduling and runtime adaptation of scientific workflow. *Concurrency and Computation: Practice and Experience*, 2009.
[21] S. Wang, D. Xuan, R. Bettati, and W. Zhao. Providing absolute differentiated services for real-time applications in static-priority scheduling networks. *IEEE/ACM Trans. Netw.*, 12(2):326–339, 2004.