Energy-efficient Multisite Offloading Policy using Markov Decision Process for Mobile Cloud Computing

Abstract

Mobile systems, such as smartphones, are becoming the primary platform of choice for user's computational needs. However, mobile device still suffer with limited resources, such as battery life and process performance. To alleviate these limitations, a popular approach used in mobile cloud computing is computation offloading, where resource-intensive mobile components are offloaded to more resourceful cloud servers. Prior researches in this area have focused on form of offloading where only a single server is considered as offloading site. Since we have an environment where mobile devices access multiple cloud providers, it is possible for mobiles to save more energy by offloading energy-intensive components to multiple cloud servers. This paper differentiates the data and computational intensive components of an application and performs a multisite offloading in a data and process-centric manners. In this paper, we present a novel model to describe the energy consumption of a multisite application execution and use the discrete time Markov chain (DTMC) to model the fading wireless channel of mobiles. We adopt Markov decision process (MDP) framework to formulate the multisite partitioning problem as a delay-constrained least-cost shortest path problem on a state transition graph. Our proposed Energy-efficient Multisite Offloading Policy (EMOP) algorithm that built on the value iteration algorithm finds the optimal solution to the multisite partitioning problem. The numerical simulations result shows that our algorithm considers the different capabilities sites to assign appropriate components among sites such that there is a lower energy cost for transferring data from mobile to cloud. A multisite offloading execution using our proposed EMOP algorithm achieved a greater reduction on the energy consumption of mobiles when compared to a single site offloading execution.

1 Introduction

Nowadays, the number of smartphone users is increasing rapidly. According to a Gartner press release from 13 February 2014 [1], worldwide sales of smartphone to end users totaled 968 million units in 2013, an increase of 42.3 percent from 2012. These numbers illustrate that smartphone are becoming the primary platform of choice for users communication and computation needs. Moreover, there are an enormous number of mobile applications available for smartphones. However, smartphones still cannot match their desktop counterparts when performing complex multimedia operations such as image and video processing, object or face recognition and augmented reality applications [2]. This is because, mobile device systems still suffer with limited resources, such as battery life, network bandwidth, storage capacity and process performance [3]. Therefore, augmenting the capabilities and prolonging the battery life of mobile devices has become one of the top research domains.

In recent years, there has been a significant amount of research performed on computation offloading [4-9]. Computation offloading has been used as a major approach in mobile cloud computing to augment mobile's capabilities by migrating computation to more resourceful computers (i.e., servers) [10]. The current cloud computing infrastructure provides mobiles with the abundance of and easy access to public cloud computing resources. Therefore, we have several cloud computing providers which uses the public clouds to solve mobile computing problems. For instance, Apple's iCloud provides a service to its customer by hosting their applications and data in public clouds (i.e., Amazon EC2 and Microsoft Azure). However, it is believed that the network latency of public clouds can be problematic for offloading the computation of perception and multimedia mobile application [27]. Thus, mobiles devices desire to access servers which have low latency for computation offloading. These servers include servers at Wi-Fi hotspot and servers at organization's private cloud [26]. Moreover, there are situations where organizations wish to store their private or proprietary data in their own private cloud. In such cases, mobile devices which manipulate these data offload computations to private cloud servers, rather than public cloud servers, for security reason and faster access of the data.

Many researchers have proposed computation offloading algorithms that increase performance and extend battery life of mobile devices by migrating the energy-intensive part of computation to server. However, most of the researches have a limited form of offloading. First, most of the schemes are limited to form of offloading where a single server is considered as the offloading site [4, 5, 9]. Since we have an environment where mobile devices access multiple cloud providers, it is possible for mobiles to offload computations to multiple servers. This approach is desirable because it assigns the appropriate application components among servers to save more energy and increase mobile performance [11]. Second, in most cases, the schemes makes offloading decision based on profiling information that assumes a stable network environment [7, 8]. However, this assumption is impractical because the mobility of users crates a dynamic bandwidth between mobiles and server. As a result, if the network profile information doesn't match the actual post-decision bandwidth, the offloading decision could lead to a critical Quality-of-Service (QoS) failure.

In this paper, we consider a real-world environment, where there are multiple heterogeneous servers for executing application components and the network bandwidth between mobiles and servers is considered to be stochastically dynamic. In this environment, a multisite offloading is implemented with a desire to perform a data-centric [8] and process-centric manner of offloading. Hence, this technique is a suitable for mobile applications consisting of both data and computational-intensive module. Real-time demanding multimedia applications, especially augment reality applications, are among some of the applications that benefit from this technique. A mobile augment reality application, which works by extracting set of features from the scene image, can be a good example for illustrating its benefits. It uses the feature descriptors to find similar looking entries in a database [12]. In this case, the feature extracting module considered as a computational-intensive module and the feature matching module as a data-intensive module. It is clear that running these modules on mobile device will consume a great amount of energy and bandwidth (Figure 1a). Therefore, it is a good approach to offload both modules to the cloud servers to save mobile energy and increase performance. So, in this scenario, multisite offloading is an efficient way of offloading

because mobiles can offload the feature extracting module to a server which is near to the mobile, and the feature matching module to a server which is near to the database [8] (Figure 1b). This approach can be considered to be optimal since it has low latency on the network which results in a higher quality of object recognition [2].

In this paper, we propose an optimal multisite offloading approach that minimizes the energy consumption of mobiles while meeting the execution deadline. Our approach differentiates the data and computational intensive components of an application and performs a multisite offloading in a data and process-centric manners. To better adapt the dynamic network bandwidth of the channels, a finite state discrete time Markov chain (DTMC) is presented to model the channel between mobiles and offloading sites [22]. In addition, we introduce a multisite partitioning algorithm which incorporates the Markov decision process (MDP) [20] to find the energy-efficient offloading decision for mobiles.

The major contributions of this paper include:

1. By combining the static analysis and dynamic profiling of an application, we introduce a novel mathematical model to describe the energy consumption of a multisite application execution in a heterogeneous environment.
2. Using the Gilbert-Elliott channel model [22] for the communication channels, we present a MDP framework to model the multisite offloading decision problem as a delay-constrained least-cost shortest path problem on a directed acyclic state transition graph.
3. In order to obtain the optimal solution for our problem, we proposes an Energy-efficient Multisite Offloading Policy (EMOP) algorithm which adopts the MDP value iteration algorithm (VIA) [20] to find the optimal offloading decision for an energy-efficient multisite application execution.

We also conduct a numerical simulation to verify the effectiveness of the multisite offloading decision. The remainder of the paper is organized as follows: In Section 2, related works is reviewed followed by system model and problem formulation in Section 3. Section 4 describes our energy-efficient multisite offloading decision under the Markovian stochastic channel. Section 5 provides the numerical simulation and evaluation of our approach. Finally, we draw our conclusions and future works in the Section 6.

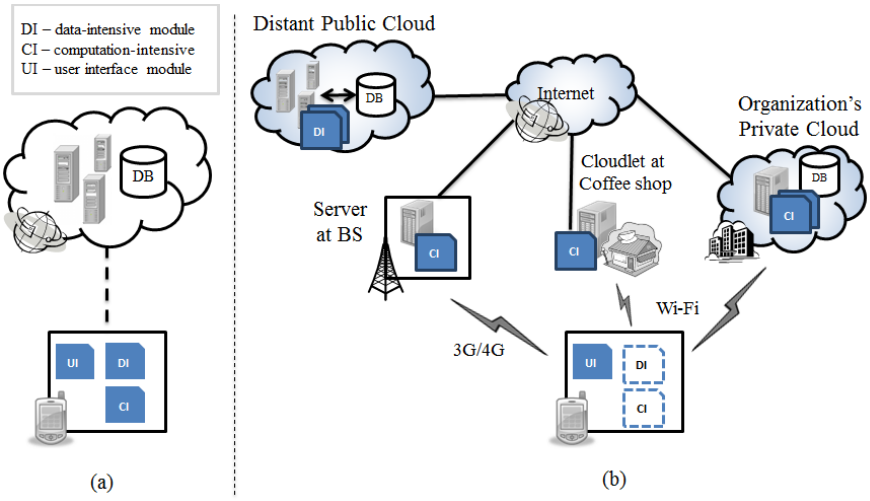


Figure 1 (a) A native way of mobile application execution. (b) A mobile application execution in multisite offloading environment.

2 Related work

The idea of transferring computation from mobiles to servers in order to augment mobiles capability and save energy has been well studied. Several methods, such as MAUI [4] and CloneCloud [5], have been proposed to support the use of augmented computation to a mobile device. MAUI uses extensive profiling to make a method level code offloading decision in Microsoft.Net runtime environment. An Integer Linear Programming (ILP) [24] approach is used to find the energy efficient deployment. Similarly, CloneCloud uses a process-based offloading methodology where the binary of the application is partitioned and are migrated to the cloud. It shows up a significant energy saving with applications such as virus scanning, image search, and behavioral profiling. But, both MAUI and CloneCloud consider a two-way partitioning algorithm that needs a continuous and extensive profiling. Our approach makes an optimal multi-way partitioning which considers multiple cloud servers. Moreover, rather than making a continuous and extensive profiling, we model the channel behavior as a stochastic process to predict the channel states.

Multisite offloading is used as a best approach for saving more energy on mobiles [11]. To the best of our knowledge, the only prior works that consider multisite partitioning is presented in [6], [7] and [8]. Niu et al. [6] present a multi-way partition algorithm, EMSO, which formulate the partitioning problem as 0-1 ILP problem. The algorithm models the program as a weighted directed acyclic graph and performs a depth-first search that traverses the search tree. Then, it computes the energy consumption of nodes under the current and critical bandwidth. The algorithm succeeds in finding the energy-efficient multisite partitioning decision, but, it doesn't provide a guarantee for completion time. Hence, it cannot adopt well for real-time demanding multimedia application. Ou et al. [7] proposed a $(K+1)$-way partitioning algorithm to keep the component interaction as small as possible. They develop a light-weight, n-way partitioning algorithm, Heavy Edge and Light Vertex Matching Algorithm (HELVM), which splits application graph to multiple servers satisfying some pre-defined constraints. Unfortunately, HELVM assumes all servers alike (i.e., homogenous capacities). Thus, Sinha and Kulkarni [8] developed a multisite offloading algorithm which treats every cloud with different capacities to adapt more practical environment where the servers have a different computational capacities and network bandwidths. In their model, they use different weights of nodes, or different network bandwidth that requires different edge weights for communication. Their work is motivated by a data-centric approach of offloading for applications that manage multiple sources of data. However, their work considers a stable channel state for making the offloading decision. In this paper, we consider heterogeneous capabilities of servers like [8] and we also consider process-centric application offloading additionally to make our target condition more practical. We analyze and predict channel states to make the offloading decision more energy efficient.

Our research is different from previous works in heterogeneity; the applications that we investigate have both data and computation-intensive components which can be offloaded over a group of distributed heterogeneous servers. Moreover, our proposed EMOP algorithm considers the heterogeneity of network access to optimize the overall offloading decision. Instead of performing a continuous extensive profiling of the channel rate, change in bandwidth rate is predicted using a Markov chain model for fading channels. Zhang et al. [9] studied the scheduling policy for collaborative execution in a stochastic channel and used LARAC algorithm to obtain the minimum energy offloading policy between a mobile device and an offloading site. However, their work is limited to a single offloading site where a steady network topology is considered. Our approach considers a multiple site application execution where change in network topology might exist. We use the MDP value iteration algorithm, which considers the multiple stochastic channels between mobile and offloading sites to obtain the energy-efficient multisite offloading decision.

3 Model Formulation

In the example we made in Section 1, we considered an augment reality application which consist several components that could be offloaded to cloud sites. The components represent the granularity of the partition which could be a method-level [4], thread-level [5] or a module [2]. The augment reality application can be formed by

several offloadable and unoffloadable components [2]. The unofflaodable components include those that handle user interaction or access local I/O devices, such as the object tracking bundles. On the other hand, the detector and recognizer/decoder bundles are considered as offloadable components. These components can migrate to multiple cloud sites without affecting the application execution.

We assume that a mobile application is represented as a sequence of components in a linear topology. This assumption is made based on that offloaded components are likely to be executed sequentially [13]. Thus, we use a weighted directed acyclic graph $G = (V, E)$ to represent the relationship among the components in mobile application. Each vertex $v \epsilon V$ denotes a component and the edge $e(u, v)$ denotes the communication channel between the component u and v. Figure 2 is an illustration of a weighted directed execution graph of a mobile application. The unshaded vertices are offloadable components while the shaded vertices are unoffloadable components due to I/O, hardware or externals constraints. The weight on the vertices, denoted by C_i for vertex i, is a vector weight representing the cost of executing the components on each site. The weight on edges, denoted by $C_{i,j}$ for edge $e(i, j)$, is also a vector weight to represents the cost of transmitting data between components in different sites. The cost metrics can either be time or energy consumption. The EMOP algorithm adopts both cost metrics to find the energy-efficient delay constrained offloading decision. These weights are constructed using both static analysis and dynamic profiling methods for an application [5].

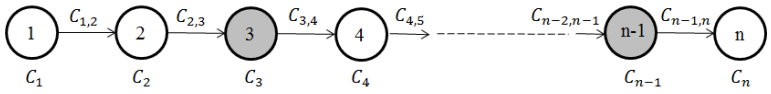


Figure 2 A mobile application represented as a weighted directed graph

3.1 Problem Formulation

In this sub-section, we present a mathematical model to describe the energy consumption models for multisite application execution, including the computational energy model for mobile execution and a transmission energy model for cloud execution. Unlike our model, some models [6, 8] formulate the energy consumption for the whole execution site rather than focusing on the energy consumption of mobiles. Since cloud servers have abundant energy compared to mobiles, our problem formulation is focused on saving the energy consumption of only mobiles.

The notations used for our models are listed in Table 1. We assume a mobile application which consist n components in a linear topology that could migrate to k offloading sites. We define the $k + 1$ execution sites as $Q = \{q_0, q_1, q_2, \dots, q_{k-1}, q_k\}$, where q_i denotes the i^{th} offloading site and q_0 represents the mobile device itself. Here, we consider every offloading site with different computational capacity and network bandwidth, thus, the energy cost on each node and edge is represented as a set of energy cost of each site. As shown in Figure 2, each component $v \epsilon V$ has an associated cost which can either be an energy cost or time cost. We define the energy cost as $E_v = \{e_{v_0}, e_{v_1}, e_{v_2}, \dots, e_{v_{k-1}}, e_{v_k}\}$, where e_{v_i} denotes the energy cost of v if it is executed on offloading site q_i. Similarly, we define $T_v = \{t_{v_0}, t_{v_1}, t_{v_2}, \dots, t_{v_{k-1}}, t_{v_k}\}$ to represent the time cost to execute component v on each offloading sites.

Table 1. Notation of multisite partitioning model

Symbols	Meaning
q_i	The i^{th} offloading site and q_0 represents the mobile device itself
e_{v_i}	The energy cost of component v if it is executed on offloading site q_i
t_{v_i}	The total time cost of component v if it is executed on offloading site q_i

$t_{v_i}^c$	The computational time of executing component v on site q_i
$t_{v_i}^s$	The time spent for sending data to the database by a component v on site q_i
$t_{v_i}^r$	The time spent for receiving data from the database by a component v on site q_i
f_i	The CPU clock speed (cycles/second) of an offloading site q_i
w_v	The total CPU cycles needed by the instructions of component v
d_{v_s}	The data (byte) sent by a component v to the database
d_{v_r}	The data (byte) received by a component v from the database
r_i	The data rate (byte/second) between site q_i and the database server
p_s	The power consumption of mobiles for sending data
p_r	The power consumption of mobiles for receiving data
p_c	The power consumption of mobiles for computing
p_{idle}	The power consumption of mobiles for being idle
$e_{u_i v_j}$	The energy cost for transferring data from component u on site q_i to component v on site q_j
$t_{u_i v_j}$	The time spent for transferring data from component u on site q_i to component v on site q_j
$d_{u,v}$	The data transferred from component u to v
$r_{i,j}$	The transmission rate between site q_i and q_j
r_g	The data is transmission rate in a good channel state
r_b	The data is transmission rate in a bad channel state

If we represent f_i as CPU clock speed (cycles/second) of an offloading site q_i and w_v as the total CPU cycles needed by the instructions of component v, then, the computational time of executing component v on site q_i can be formulated as:

$$t_{v_i}^c = \frac{w_v}{f_i}, \forall v \in V \text{ and } \forall i \in [0, k] \quad (1)$$

Our model also considers the time cost incurred due to database or other remote data access by component. We denote the data (byte) sent and received by a component v as d_{v_s} and d_{v_r}, respectively. Without loss of generality, we assume that components access data from a single database server. Thus, if we represent r_i as the data rate (byte/second) between site q_i and the database server, the communication time spent for sending and receiving data from the database by a component v on site q_i is given as:

$$t_{v_i}^s = \frac{d_{v_s}}{r_i} \text{ and } t_{v_i}^r = \frac{d_{v_r}}{r_i}, \forall v \in V \text{ and } \forall i \in [0, k] \quad (2)$$

Note that the size of data for data-intensive component is considered to be large. Thus, the data-intensive components have a higher data access time compared to the computational-intensive components. On the other hand, computational-intensive components have large number of instruction, hence, higher computational time. We summarize the total time cost of a component v on site q_i as:

$$t_{v_i} = t_{v_i}^c + t_{v_i}^s + t_{v_i}^r \quad (3)$$

Likewise, the energy consumption E_v of a component is calculated as the amount of energy a mobile spends while executing the component or while waiting for the component to be executed on offloading sites. So, if we assume the transmission power of mobile devices is fixed, the energy cost of a component is formulated as:

$$e_{v_i} = \begin{cases} t_{v_i}^c \times p_c + t_{v_i}^s \times p_s + t_{v_i}^r \times p_r, & \forall v \in V \text{ and } i = 0 \\ t_{v_i} \times p_{idle}, & \forall v \in V \text{ and } \forall i \in [1, k] \end{cases} \quad (4)$$

where p_s and p_r represent the power consumption of mobiles for sending and receiving data, respectively. p_c is the power consumption of mobiles for computing and p_{idle} for idle. Note that, for unoffloadable components, the energy cost, e_{v_i}, is assigned infinity for all offloading sites except the mobile.

In addition to the vertex costs, the communication energy cost on edge $e(u,v)$ is represented as $e_{u,v} = \{e_{u_0v_0}, e_{u_0v_1}, \dots, e_{u_0v_k}, e_{u_1v_0}, \dots, e_{u_kv_k}\}$, where $e_{u_iv_j}$ denotes the cost of edge $e(u,v)$ if component u is executed on site q_i and component v on site q_j. In the same way, we define $t_{u,v} = \{t_{u_0v_0}, t_{u_0v_1}, \dots, t_{u_0v_k}, t_{u_1v_0}, \dots, t_{u_kv_k}\}$ to represent the communication time spent on edge $e(u,v)$ for transferring data from u to v, where $t_{u_iv_j}$ represents the time spent during transferring data from component u on site q_i to component v on site q_j. The communication time of edge $e(u,v)$ depends on the byte of data transferred and the network bandwidth used between the sites [11]; hence, it is given as:

$$t_{u_iv_j} = \frac{d_{u,v}}{r_{i,j}} \ , \forall (u,v) \in E \ and \ \forall i,j \in [0,k] \tag{5}$$

where $d_{u,v}$ denotes the transferred data from component u to v, and $r_{i,j}$ denotes the transmission rate between site q_i and q_j. Here, we assume that the components of the application are already replicated on each site; therefore, there is no need of transferring the component itself. In addition, we assume that, there is no transmission power for components in the same execution site. Thus, the communication energy cost is given as:

$$e_{u_iv_j} = \begin{cases} t_{u_iv_j} \times p_s \ , & \forall (u,v) \in E \ and \ i = 0, j \in [1,k] \\ t_{u_iv_j} \times p_r \ , & \forall (u,v) \in E \ and \ i \in [1,k], j = 0 \\ t_{u_iv_j} \times p_{idle} \ , & \forall (u,v) \in E \ and \ i,j \in [1,k], i \neq j \\ 0 & otherwise \end{cases} \tag{6}$$

The first and second formulas represents the communication energy spent on edge for sending data from mobile to offloading site and receiving data from offloading site to mobile, respectively. The third formula represents the energy a mobile spent while waiting data transfer between components on different offloading sites. Based on the measurement results found in [11], we made an assumption that $p_s > p_r > p_c > p_{idle}$.

Our objective is to find an offloading decision that assigns n components to $k+1$ sites such that the whole execution of the graph has minimum energy consumption at mobile with acceptable execution delay. This problem can be modeled as an energy optimization problem with a constraint in completion time. A standard integer linear programming (ILP) solver has been successful to solve such kind of optimization problem [5, 24], therefore we formulate 0-1 ILP optimization problem that adopts the context of multisite offloading. The formulated minimization problem is given as follows:

$$\text{minimize } \{Energy(G) = \textstyle\sum_{v \in V} \sum_{i=0}^{k} (e_{v_i} \times \alpha_{v_i}) + \sum_{(u,v) \in E} \sum_{i=0}^{k} \sum_{j=0}^{k} (e_{u_iv_j} \times \alpha_{u_i} \times \alpha_{v_j})\} \tag{7}$$

$$s.t. \quad \textstyle\sum_{v \in V} \sum_{i=0}^{k} (t_{v_i} \times \alpha_{v_i}) + \sum_{(u,v) \in E} \sum_{i=0}^{k} \sum_{j=0}^{k} (t_{u_iv_j} \times \alpha_{u_i} \times \alpha_{v_j}) \leq \Delta_{delay} \tag{1}$$

$$\forall v: \textstyle\sum_{i=0}^{k} \alpha_{v_i} = 1 \tag{2}$$

Here, we define α_{v_i} to be the decision variable that is equal to 1 if component v is executed on offloading site q_i and 0 otherwise. The optimization solver solves for the optimal assignment of α_{v_i} which minimizes the total energy consumption of mobiles. The first constraint stipulates that the total execution time of the application must be below the acceptable delay Δ_{delay}. The second constraint is introduced to enforce that each component is assigned to exactly one offloading sites. Note that the optimization problem is solved under some expectation of the channel state. As a result, it is challenging to solve an optimal offloading decision for unstable channels.

The optimization problem is similar to that of partitioning a finite element graph into a certain number of disjoint subsets of vertices while fulfilling some given objective [7]. However, this type of graph partitioning problem is known to be NP-complete [15, 16], more specifically NP-hard in case of multisite partitioning. Thus, we are not able to solve the optimal offloading decision for any application except small applications with small graphs [8]. A general approach to deal with NP-complete problem is to look for a polynomial-time algorithm that guarantees finding an approximate solution to the optimal one [17]. In this paper, we use a novel approach to transform the optimization problem to stochastic shortest path problem on a directed acyclic graph, then, we present an efficient algorithm which finds an approximate solution to optimization problem in polynomial time. Before describing our algorithm, we first show how we can model the dynamic characteristics of the channels as a finite-state Markovian model.

3.2 Finite-state Markov Channel Model

For the purpose of computation offloading, mobile devices create a wireless connection with several sites through a number of access points as shown in Figure 1b. As a result, the transmission rate of the connection varies depending on user's location and time. Because, in computation offloading, the execution time heavily depends on the cost of transfer input/output data over the wireless link, the data transmission rate will have significant impact on offloading decisions [13].

The Markovian chain model has proved to be a very effective mathematical tool to describe a wireless channels where transmission rates varies stochastically over time. We consider a finite-state Markov model for each wireless communication between mobile and offloading sites. This model is a simple and effective approach which is suited to approximate fading channel models [18]. Although there are a lots of Markovian chain based models, we consider the Gilbert-Elliott channel to model the communication link since it is used to refer to wide class of finite-state fading channels. This channel model is a Markov chain with two states: good (denoted by g) or bad (denoted by b). If the channel is good, data is transmitted with a high rate (r_g), otherwise if the channel is bad, it sends data with a low rate (r_b) [19]. The transition between the two channel states occurs in discrete time instants, so that the channel is assumed to stay in a given state for some time unit. Let $\gamma_n = g$ if the channel is good during the n^{th} time unit, and $\gamma_n = b$ otherwise. Hence, the state transition matrix of the channels, *P*, is given by:

$$P = \begin{bmatrix} p[\gamma_n = g|\gamma_{n-1} = g] & p[\gamma_n = b|\gamma_{n-1} = g] \\ p[\gamma_n = g|\gamma_{n-1} = b] & p[\gamma_n = b|\gamma_{n-1} = b] \end{bmatrix} = \begin{bmatrix} p_{gg} & p_{gb} \\ p_{bg} & p_{bb} \end{bmatrix}$$

The steady-state distribution is denoted as $\mu = [\mu_g, \mu_b]$, where μ_g and μ_b denotes the steady-state probability of the channel being in good state and bad state, respectively. The steady-state probability of both states is given as:

$$\mu_g = \frac{p_{bg}}{p_{bg} + p_{gb}} \quad , \quad \mu_b = \frac{p_{gb}}{p_{bg} + p_{gb}}$$

Accordingly, if we assume the transmission power of mobiles is static, the expected steady-state transmission rate *R* can be calculated as $R = \mu_g r_g + \mu_b r_b$.

3.3 Markov Decision Process formulation

Under the stochastic channel model, we now describe how to formulate the multisite offloading decision problem as a Markov decision process. The MDP model consists of the following five elements: decision epochs, states, actions, transition probabilities, and costs [20].

We consider a finite horizon discrete time problem, where decisions are made at the beginning of a period or stage. The decision epochs represent a point of time for decision. In our formulation, we formed each stage based on the number of components in the application. We introduce two new nodes in the execution graph to represent the

starting and ending of the execution. As a result, we have $n+2$ components in the application execution graph forming $n+2$ decision epochs. We represent the decision epoch as $T = \{0,1,2,\dots,n,n+1\}$ where decision epoch $t \in T$ indicates that component t has already executed.

The decision maker chooses an action based on the system state information, denoted by S. Each state $s \in S$ is characterized by the combination of the channel states and the execution location of a component. We define the system state at a decision epoch t as $x_{t,i} = (t, i, \gamma_t)$, where γ_t denotes the channel state for the next epoch between mobile and offloading sites (i.e., either g or b), and $i \in [0, k]$ denotes the location of the executed component t. Figure 3 illustrates the state transition of the system under the Markovian stochastic channel. Because, we assume the initial channel state to be good, and executions start and end at the mobile, the initial and final system state is given as $x_{0,0} = (0,0,\gamma_0 = g)$ and $x_{n+1,0} = (n+1,0,\gamma_{n+1})$, respectively.

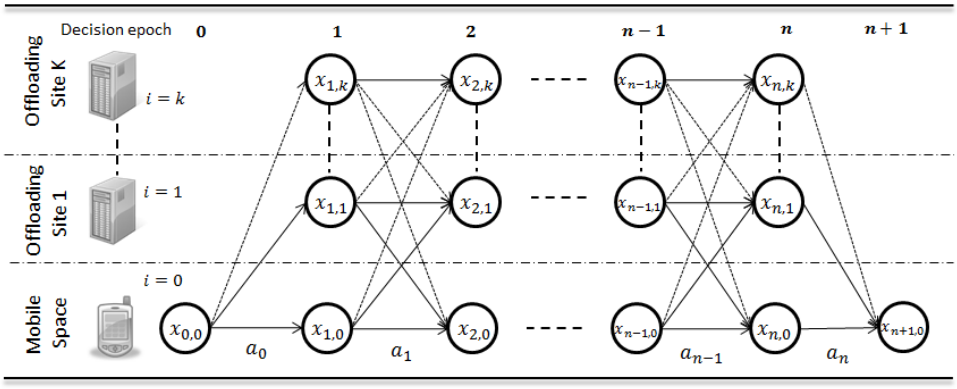


Figure 3 State transition of the system

Given the current state, the decision maker can choose among two major actions: migrate execution or continue execution. Migrate execution action executes the next component in a different execution site whereas continue execution action keeps executing the next component locally. Here, because we assume that the components of the application are already replicated on each site, migrate execution action performs only the action of sending the input data to a component in another site. Thus, at each stage, for $K+1$ sites we have $K+1$ number of action choices. We define the action set for state s as A_s, and the action taken at stage t is represented by $a_t \in A_s$. Based on the chosen action a in the state s, the state transition probability function for the next state s' is given by $P[s'|s,a]$. This function applies the Markov property since the next state of a system is predicted from the current state only. Note that the transitional probabilities of the channels is not affected by the action, therefore, we consider the transition probability in the system equals to the transition probability of the channel state [9].

The decision rule prescribes a procedure for action selection in each state at a specified decision epoch. This paper considers a stochastic Markovian decision rule because the state channel between mobile and offloading sites are considered to be stochastic. A decision rule $\delta_t: S \to A$ is a mapping from states to action at decision epoch t. It indicates which action to choose when system is at a specific state at a given time. A policy $\pi = (\delta_0, \delta_1, \dots, \delta_n)$ represents a sequence of decision rule to be used at all decision epoch. A policy is said to be stationary if it forms $\pi = (\delta, \delta, \dots)$ or $\delta_t = \delta$ for all t. For convenience, we let $\pi = (\pi(0), \pi(1), \dots)$, where $\pi(t)$ denotes the action to take at decision epoch t under policy π. It follows that if a stationary policy is employed, then the sequence of states $\{X_t, t = 0,1,\dots,n+1\}$ forms a Markov chain with transition probability $P_t(s'|s,\pi(s))$. According to [21, Thm 2.2], there exist a stationary policy π^* which is optimal for all policy. Therefore, in this paper, our goal is to find an optimal stationary policy that suggests the best action which minimizes the sum of the cost incurred at the current stage and the least total expected cost that can be incurred from all subsequent stages. We

denote $V^{\pi}(s)$ to be the expected total cost of executing the application given the initial state s and a policy π. $V^{\pi}(s)$ is calculated as

$$V^{\pi}(s) = E^{\pi}[\sum_{t=0}^{n} C(X_t, a_t)|X_0 = s] \quad (8)$$

where E^{π} represents the conditional expectation with respect to policy π and $C(X_t, a_t)$ is the cost incurred at stage t by taking action a_t. In this paper, the cost function is considered to be either the amount of energy consumed or time spent at the mobile by taking the specific action. In the remainder of the paper, we will see how we can calculate and find the optimal policy which minimizes the total expected cost.

4 Energy-efficient Multisite Offloading Decision

In this section, we propose a novel algorithm that considers the stochastic behavior of the channels to solve the approximate multisite offloading decision. Under the Markovian stochastic channel, we adopt the MDP framework to predict the future channel states and make the energy-efficient offloading decision. This section introduces the optimality equations and the VIA then shows how we can use the algorithms to find the energy-efficient multisite offloading decision.

We target to find an optimal multisite decision which minimize the energy consumption of mobiles and satisfy the required execution deadline. Therefore, we represent the cost function in Section 3 for both time and energy consumptions. If we assume the transition of state occurs from $X_t = (t, i, \gamma_t)$ to $X_{t+1} = (t+1, j, \gamma_{t+1})$, then, the energy cost function, C_E, and time cost function, C_T, is given as follows:

$$C_E(X_t, a_t) = \sum_{X_{t+1}} P_t(X_{t+1}|X_t, a_t)[e_{(t+1)_j} + e_{t_{i(t+1)_j}}(\gamma_t)] \quad (9)$$

$$C_T(X_t, a_t) = \sum_{X_{t+1}} P_t(X_{t+1}|X_t, a_t)[t_{(t+1)_j} + t_{t_{i(t+1)_j}}(\gamma_t)] \quad (10)$$

Here, $e_{t_{i(t+1)_j}}(\gamma_t)$ and $t_{t_{i(t+1)_j}}(\gamma_t)$ are calculated using the bandwidth rate between site q_i and q_j under the channel state of γ_t. These cost functions are important parameters for calculating the optimality equation of both energy and time of the multisite application execution. We describe the generalized optimality equation in the following sub section.

4.1 Optimality Equations and Value Iteration Algorithm

Under the established MDP model, we use the Bellman's equation [21] to express the optimality condition. We consider the function $Q(s, a)$ to be the expected cost of taking an action a on state s and, thereafter, act optimally. Then, the Bellman equation for our context takes the form:

$$Q(s, a) = \begin{cases} 0 & if\ s \in X_{n+1} \\ C(s, a) + \sum_{s'} P(s'|s, a)V(s') & otherwise \end{cases} \quad (11)$$

Here, we consider X_{n+1} to be the goal state, thus, the cost of taking any action on the goal state is zero. Moreover, we denote $V(s)$ to be the minimum expected total cost given initial state s, i.e.,$V(s) = \min_{\pi} V^{\pi}(s)$. Thus, we represent the optimality equation $V(s)$ as:

$$V(s) = min_a[Q(s, a)] \quad (12)$$

The solution of the optimality equation represents the minimum expected total cost and the MDP optimal policy π^*. Note that the MDP optimal policy indicates the decision as to which site to migrate the execution, given the current state. Algorithms; such as value iteration algorithm (VIA), policy iteration algorithm (PIA), and linear programming, can be applied to solve the Bellman's optimality equation [21]. We used VIA in our work due to its theoretical

simplicity and easiness in coding and implementation [20]. The VIA which yields the optimal stationary policy and the corresponding expected total cost to our optimization problem is given as follows:

Value iteration algorithm (VIA)

1. Initialize $V_0(s) = 0$ for all $s \in S$
2. **For** $k = 1$ to $n + 1$
3. **For** each state $s \in$ S:
4. **For** each action $a \in \mathrm{A}_s$
5. $Q_k(s,a) = C(s,a) + \sum_{s'} P(s'|s,a)V_{k-1}(s')$
6. $\pi_k^*(s) = arg\ min_a[Q_k(s,a)]$
7. $V_k(s) = Q_k\big(s, \pi_k^*(s)\big)$
8. **Return** $< Q_{n+1}, V_{n+1}, \pi_{n+1}^* >$

Note that the algorithm reaches the goal state with a maximum of $n + 1$ steps. This indicates that the value function doesn't improve after $n + 1$ iterations. Therefore, the algorithm terminates after the $n + 1$ iteration by returning the optimal stationary policy and value function. The value function at the last iteration, $V_{n+1}(s)$, is considered as $V(s)$. Once the optimal policy is returned, it can be stored in a matrix format, where each entry identifies the optimal action which represents the offloading decision for the given state.

4.2 The Energy-efficient Multisite Offloading Policy Algorithm

In this section, we describe an energy-efficient multisite offloading policy (EMOP) algorithm which minimizes the energy consumption of mobile and satisfies the execution deadline constraint. We show how we can use the policy generated from the VIA to find the optimal decision for an energy-efficient multisite execution. The EMOP algorithm takes the current state as an input and generates the optimal offloading decision for every possible future state.

Our optimization problem can be considered to be similar to delay-constrained least-cost path problem on state transition graph from $x_{0,0}$ to $x_{n+1,0}$. Such type of restricted shortest path (RSP) problems can be solved using several optimization algorithms, such as Back-Forward heuristic algorithm [17] and Lagrangian-based linear composition algorithm [25]. These algorithms work first by determining the least cost path and the least delay path from every node to destination on the graph. Then, the Back-Forward algorithm searches the graph in two segments and finds the optimal path. The Lagrangian-based algorithm combines the delay and cost of each path, and finds the approximate optimal path for the composite value. Similarly, the EMOP algorithm starts by calculating the least energy and time cost of every state node using the VIA. However, the EMOP algorithm, then, uses a novel approach to explore every state to find the optimal offloading decision which is energy-efficient. The energy-efficient offloading decision indicates the best action that minimizes the expected energy cost and meets the deadline. Rather than setting the offloading decision of every component at the beginning of an application execution as shown in [6, 8], the EMOP algorithm provides the optimal decision for each component execution under all the possible system states. Thus, the optimal offloading decision of a component is made during application execution based on the observed channel state and the current execution location. This decision is represented by the energy-efficient offloading policy function which maps the possible states to the optimal offloading decision.

Algorithm 1: EMOP algorithm

Input: $S, A, P, C_E, C_T, \Delta_{delay}, s_0$

Output: optimal offloading policy ρ

1. $< Q_E, V_E, \pi_E^* > \leftarrow VIA(S, A, P, C_E, s_0)$ //finds the optimal energy and time cost for each state
2. $< Q_T, V_T, \pi_T^* > \leftarrow VIA(S, A, P, C_T, s_0)$ //at the same time finds the optimal policy
3. **if** $V_T(s_0) > \Delta_{delay}$
4. **then** return "no feasible solution"
5. **For** each state s in S:
6. **do** $a_E \leftarrow \pi_E^*(s)$
7. **if** $a_E = \pi_T^*(s)$ **or** $Q_T(s, a_E) \leq \Delta_{delay}$
8. **then** $\rho \leftarrow \rho + \{(s, a_E)\}$ //consider the action a_E as optimal
9. **continue**
10. **do** $Q_E \leftarrow Q_E - \{(s, a_E)\}$
11. **if** $Q_E = \emptyset$
12. **then** $a_E \leftarrow \pi_E^*(s)$
13. **break**
14. **else**
15. $a_E \leftarrow arg\ min_a[Q_E(s, a)]$ //find the next energy optimal action
16. **Until** $Q_T(s, a_E) > \Delta_{delay}$
17. $\rho \leftarrow \rho + \{(s, a_E)\}$
18. **return** ρ

Algorithm 1 shows the EMOP algorithm for constructing the energy-efficient offloading policy of multisite execution. We use a subscript of E and T on the MDP policies and cost functions to represent the energy and time, respectively. The algorithm explores every state node and selects the action with the lowest energy and acceptable delay to be the optimal action. If the action with lowest energy doesn't satisfy the delay constraint, the algorithm checks the next lowest energy action. It performs this operation until it finds an action which satisfies the delay constraint. After traversing the whole states, the EMOP algorithm gets the optimal multisite offloading policy which is used to make the optimal decision for every future state the system encounters.

The EMOP algorithm finds the optimal offloading decision with a computational complexity of $O(SA)$, where S is the state space, and A is the action set. This computation complexity didn't reflect the complexity of VIA. Although, the EMOP algorithm can be considered as a computational expensive operation for resource constrained mobiles, an assumption could be made so that the calculation for the optimal decision is performed by the remote server before the beginning of an application execution [2]. Therefore, the mobile devices only store the matrix form of the returned optimal function. The offloading decision will stay valid for the whole execution of the application under the current network environment.

5 Numerical Simulation and Evaluation

To evaluate the performance of our proposed EMOP algorithm in terms of energy saving, we conduct numerical simulation. We compare the results with a single site offloading execution. We also discuss the decision characteristics of the EMOP algorithm in different scenario, since the algorithm decision relies on the different component types.

5.1 Simulation Setup

For our simulation, we consider a scenario where there are three offloading sites (i.e., K=3), as in the heterogeneous system shown in Figure 1b. Site 1 simulates a server which is near to the mobile. Site 2 and Site 3 simulates cloud servers with different characteristics. Site 2 is considered to be the database server and Site 3 is a cloud server with higher computational capacity. We assume a two-state stochastic channel between mobile and offloading sites with

a small threshold on the state transition probability. Therefore, we consider the state transition probabilities of the channel to be $p_{gg} = 0.995$ and $p_{bb} = 0.96$ [9]. For simplicity, the channel between the three offloading sites is assumed to be a deterministic channel with bandwidth rate which is ten times faster than the bandwidth of the mobile. Since, the database exists at Site 2, data intensive components on mobile devices and other offloading sites have to access the database with the same bandwidth rate of Site 2. We make additional assumptions on the power consumption rates of mobiles from the power readings of HP iPAQ PDA in [11]. The summarized system parameters used for our simulations are listed in Table 2 as follows.

Table 2. Simulation parameters of testing machines

Machine	Parameters
Mobile	$f_0 = 500MHZ$ $p_s = 1.3W, p_r = 1.0W,$ $p_c = 0.9W, p_{idle} = 0.3W$
Site 1	$f_1 = 2GHZ$ $r_g = 100kb/s, r_b = 50kb/\text{s}$
Site 2	$f_2 = 3GHZ$ $r_g = 50kb/s, r_b = 10kb/\text{s}$
Site 3	$f_3 = 5GHZ$ $r_g = 50kb/s, r_b = 10kb/\text{s}$

To represent real-world scenarios, we generate three types of application graphs using a certain schemes. The graphs are generated randomly by a combination of data-intensive (DI) and computational-intensive (CI) components. The first application we build for the simulation is a CI application, similar to N-queens puzzle and Sudoku solver application [23]. Since these types of applications are computationally intensive, we make CI components occupy 80% of the application graph and the rest is randomly selected from others. Similarly, to simulate the DI mobile application, like virus scanning application [23], DI components are used to fill 80% of the second application graph. Finally, we build a third application graph using a random distribution of DI & CI components. The parameters used to generate the graph components are presented in Table 3. Note that the weight on each node and edge is chosen from the given range.

Table 3. Configuration for generating application graph

Configuration	Parameter		Value
Computational-Intensive (CI)	Node weight	w_v (M cycles)	550-650
		d_{v_s}/d_{v_r} (KB)	4-8
Data-Intensive (DI)	Node weight	w_v (M cycles)	100-200
		d_{v_s} (KB)	25-30
		d_{v_r} (KB)	15-20
	Edge weight	$d_{u,v}$ (KB)	100-120

5.2 Simulation Results

First, we measured the energy consumption of a multisite execution using EMOP algorithm, and a single site execution. Then, we compared them and discuss the decision characteristics of the EMOP algorithm for different scenarios.

5.2.1 Energy consumption evaluation

The energy consumption of application for single site execution, and multiple sites execution using the decision of EMOP algorithm are presented in Table 4. In this simulation, we assumed every node to be an offloadable component even though application execution starts and ends at mobile device. We consider the steady-state transmission rate for the channel between mobiles and sites. Thus, the energy cost (i.e., consumption) of a single site execution is calculated by totaling the weights on the edges and nodes of the graph corresponding to a specific site. Similarly, for multisite execution, the energy cost is calculated by totaling the weights on the edges and nodes of the graph based on the offloading decision of the EMOP algorithm.

From the simulation result in Table 4, we make two observations. First, when the number of nodes in the application graph are small, the energy consumption of executing an application on server which is near to the mobile (i.e., Site 1) results in larger energy saving compared with the cloud server execution (i.e., Site 2 and Site 3). However, as the number of nodes increases, the energy consumption of executing on server, near to the mobile, increases and finally outweighs the cloud server execution. This occurs because the energy wasted by application input/output data transfer becomes insignificant in the overall energy cost. Thus, since cloud servers have a higher computation capacity, they can save more energy than the servers which are near to the mobile.

Our second observation is about the multisite execution. The multisite execution, using the EMOP algorithm, seeks the energy-efficient execution of components on each site. The algorithm decides to offloads most of the components to the cloud servers where there is a higher computational speed and faster access to a database. Hence, the multisite execution consumes the least energy and achieves more energy saving as much as 30.8% compared to single site executions. When the number of nodes in the application graph are small, the multisite execution results in larger energy saving compared to single site execution, particularly cloud server executions (i.e., Site 2 and Site 3). The reason is that the EMOP algorithm makes the initial offloading decision to the higher bandwidth server, thus, eliminating the energy cost incurred due to data transfer on lower bandwidth of cloud servers. However, as the number of nodes increase, the difference between the amounts of energy saved by the multisite execution and cloud server execution gradually decreases. This occurs because the EMOP algorithm decides to offloads most of the application components to a single cloud server, hence, creating a smaller difference with a single site execution. Nevertheless, as the number of nodes increase, the energy saved by the multisite execution still increases when compared to a single site execution on server which is near to the mobile.

Table 4. Energy consumption comparison for different application graph executing on different sites

Node	Graph	Energy(j)			
		Site 1	Site 2	Site 3	Multisite
10	CI	6.39	8.56	7.93	5.49
	DI	4.77	7.55	7.74	4.10
	Random	5.48	7.75	7.64	4.93
30	CI	12.02	12.24	10.64	7.65
	DI	7.31	8.47	8.93	5.52
	Random	8.87	9.51	9.27	6.59
60	CI	20.25	17.21	13.97	11.32
	DI	11.21	10.22	11.28	7.29

	Random	15.19	12.99	12.62	9.57
100	CI	33.53	25.54	19.45	16.84
	DI	16.99	13.21	14.93	9.75
	Random	24.64	18.07	17.14	14.40
150	CI	47.00	34.01	25.40	22.93
	DI	22.77	15.41	18.23	12.36
	Random	35.24	24.30	22.44	19.31

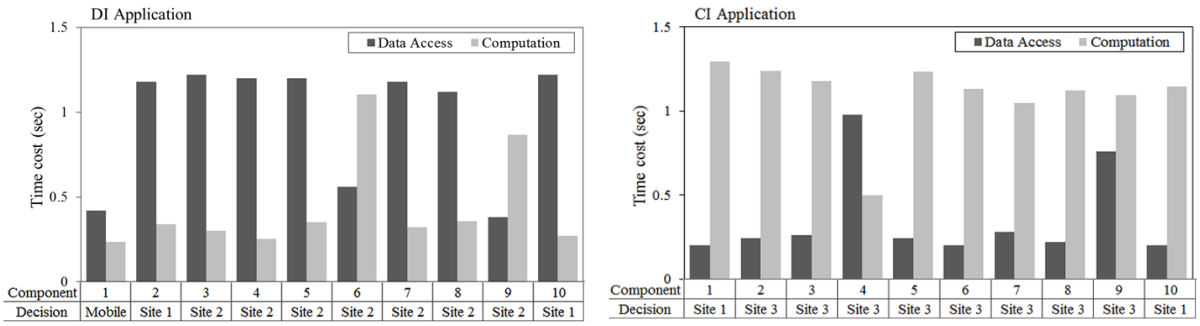
The execution with the EMOP algorithm can be performed with a small delay. Table 5 shows the energy consumption against time delay for application execution in different execution locations. The results are from the execution of a random application graph with 100 nodes. We observe that the multisite execution, using the EMOP algorithm, maintains the optimal energy under all delay constraints. In addition, we observe that only the multisite execution is possible for the delays which are smaller than ten seconds. The reason is that the multisite execution tends to make the first offloading to the higher bandwidth server which is near to the mobile. Then, the multisite executes the remaining components on the cloud servers where there is a higher computational speed and faster access to database. However, the energy cost gradually increase for lower delay constraints because sometimes the algorithm is forced to selects the least delay decision rather than the least energy decision.

Table 5. Energy consumption against time delay for different execution sites

Δ_{delay} (sec)	**Energy (j)**			
	Multisite	**Site 1**	**Site 2**	**Site 3**
18	14.40	24.64	18.07	17.14
16	14.40	24.64	18.07	17.14
14	14.40		18.07	17.14
12	14.40		18.07	
10	14.40			
9	18.63			
8	23.25			

5.2.2 *Evaluation of the multisite offloading decision*

To evaluate the decision of the EMOP algorithm, we perform two simulations in different scenarios. In the first simulation, our objective is to examine the decision of the algorithm for different component types. Thus, we consider an application graph with a light edge weight so that the decision of the algorithm mainly depends on the weight of the nodes (i.e., the type of the components). In this case, we can have an energy efficient multisite execution if we execute DI components on database server and CI components on higher computational speed server. This approach is considered to be optimal since it has low energy cost for computation and accessing the database. The simulation result in Figure 4 shows that the EMOP algorithm considers this approach for making the energy efficient offloading decision. We observe that DI application execute most of its components on cloud server where the database exists (Figure 4a) and CI application executes most of its components on higher computational speed cloud server (Figure 4b). However, in both cases, we observe that the algorithm does not make a direct migration from mobile to cloud servers. Instead, it makes the initial and final migration to a server which is near to mobile. This occurs because the server has a low latency on bandwidth which results in low energy cost for migration.

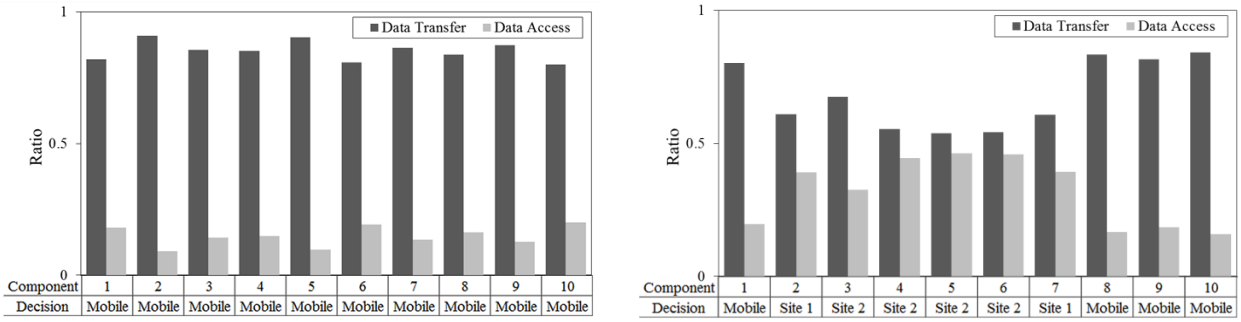


(a) (b)

Figure 4 Multisite offloading decision under light edge weight

Minimizing the energy cost on data access and data transfer of components is an important factor for saving energy. Therefore, we perform another simulation in order to examine the decision characteristics of the EMOP algorithm for different size on data access and data transfer of components. The simulation is performed using a DI application graph having a variable data size on its edges. Thus, we modify the simulation parameter of the edge data weight to be in a range of 30-350KB. Figure 5 shows the decision of the algorithm for different ratio on data weight of edge and nodes of a graph. We make two observations from the results. First, when there is high ratio of data transfer between components, the algorithm decides to keep the computation at the mobile (Figure 5a). It is because the energy cost of transferring input and output data is expensive compared to the total cost of local execution. Second, for some threshold on the ratio, the algorithm decides to offloads the execution to a server which is near to the mobile (Figure 5b). This occurs when the energy cost of input and output data transfer is small compared to the data access cost of the nodes on mobile. Furthermore, it is also observe that the EMOP algorithm selects the optimal point to make the transition from the server back to the mobile.

Based on our simulation results, we can observe that there is at most one migration between sites, and between mobile and site. This observation supports the one time offloading property in [9, 13], where at most one migration from mobile device to the cloud occurs. Our observation, additionally, suggests the existence of one time offloading property between sites.



(a) (b)

Figure 5 Multisite offloading decision under different edge and node weights

6 Conclusion and Future work

Since mobile devices access multiple cloud providers, multisite computation offloading is practicable. However, most of the current offloading schemes doesn't consider the different capabilities the multiple sites and make

computation offloading to a single server. Thus, they can only save a limited amount of mobile energy. In this paper, we proposed a multisite offloading policy for mobile devices which considers the heterogeneity of offloading sites to save more energy. The offloading policy assigns the appropriate components between mobile and sites to have minimized energy consumption at mobile. We built a mathematical model to describe the energy consumption of multisite application execution which considers the data-intensive and computational-intensive components of an application. Because the dynamic bandwidth of the wireless channel of mobiles is a critical factor on the offloading decision, we adopted discrete time Markov chain to model the fading channel. We applied the Markov decision process framework to formulate the multisite decision problem as a delay-constrained least-cost shortest path problem on acyclic state transition graph. A Markov cost model is presented in this paper to appropriately formulate the energy and time consumption of offloading decisions. Finally, we proposed an efficient algorithm that considers the dynamic characteristics of the channels, and finds an approximate solution to optimization problem. The proposed EMOP algorithm incorporates the value iteration algorithm to find the energy-efficient multisite offloading policy for Markov chain model. Therefore, mobile applications which consists a data-intensive and computational-intensive components can save more energy by considering a multisite execution using the EMOP algorithm. Moreover, the multisite offloading policy is used to have the energy-efficient offloading decision in all channel states. We performed several simulations to verify the performance the algorithm. The results show that the EMOP algorithm is an efficient multisite computation offloading approach for mobile devices and outperforms the mobile and single site executions with respect to both energy consumption and execution time. In addition, our observation suggests the existence of one time offloading property between sites.

For future researches, we intend to improve the algorithm to allow more precise offloading by considering many structures of components. For instance, our current work doesn't differentiate the data structure of component which may force different data structure components to be offloaded at the same site. In addition, we will characterize the ratio of data access and data transfer by components to incorporate it in EMOP algorithm. This will give us a better perspective for making an optimal offloading decision. Furthermore, we need to implement the algorithm in real-life scenarios to evaluate the efficiency and performance of the algorithm.

References

[1] http://www.gartner.com/newsroom/id/2665715

[2]Tim Verbelen, Tim Stevens, Pieter Simoens, Filip De Turck, and Bart Dhoedt. "Dynamic deployment and quality adaptation for mobile augmented reality applications." Journal of Systems and Software 84, no. 11: 1871-1882 (2011).

[3]Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. "A survey of computation offloading for mobile systems." Mobile Networks and Applications 18, no. 1: 129-140 (2013).

[4]Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. "MAUI: making smartphones last longer with code offload." In Proceedings of the 8th international conference on Mobile systems, applications, and services, pp. 49-62. ACM, (2010).

[5]Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud." In Proceedings of the sixth conference on Computer systems, pp. 301-314. ACM, (2011).

[6] Ruifang Niu, Wenfang Song, and Yong Liu. "An Energy-Efficient Multisite Offloading Algorithm for Mobile Devices." International Journal of Distributed Sensor Networks 2013 (2013).

[7]Shumao Ou, Kun Yang, and Jie Zhang. "An effective offloading middleware for pervasive services on mobile devices." Pervasive and Mobile Computing 3, no. 4: 362-385 (2007).

[8]Kanad Sinha, and Milind Kulkarni. "Techniques for fine-grained, multi-site computation offloading." In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 184-194. IEEE Computer Society, (2011).

[9]Weiwen Zhang, Yonggang Wen, and Dapeng Oliver Wu. "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing." In INFOCOM, 2013 Proceedings IEEE, pp. 190-194. IEEE, (2013).

[10]Zhiyuan Li, Cheng Wang, and Rong Xu. "Computation offloading to save energy on handheld devices: a partition scheme." In Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems, pp. 238-246. ACM, (2001).

[11]Karthik Kumar, and Yung-Hsiang Lu. "Cloud computing for mobile users: Can offloading computation save energy?" Computer 43, no. 4: 51-56 (2010).

[12] Kiryong Ha, Padmanabhan Pillai, Grace Lewis, Soumya Simanta, Sarah Clinch, Nigel Davies, and Mahadev Satyanarayanan. "The impact of mobile multimedia applications on data center consolidation." In Cloud Engineering (IC2E), 2013 IEEE International Conference on, pp. 166-176. IEEE, 2013.

[13] Yuan Zhang, Hao Liu, Lei Jiao, and Xiaoming Fu. "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing." In Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on, pp. 80-86. IEEE (2012).

[14] Paolo Di Lorenzo, Sergio Barbarossa, and Stefania Sardellitti. "Joint Optimization of Radio Resources and Code Partitioning in Mobile Cloud Computing." arXiv preprint arXiv: 1307.3835 (2013).

[15] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojicic. "Adaptive offloading for pervasive computing." Pervasive Computing, IEEE 3, no. 3: 66-73(2004).

[16] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, (1979).

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms. Vol. 2. Cambridge: MIT press, (2001).

[18] Fulvio Babich, and Giancarlo Lombardi. "A Markov model for the mobile propagation channel." Vehicular Technology, IEEE Transactions on 49, no. 1: 63-73(2000).

[19]Yanting Wu, and Bhaskar Krishnamachari. "Online learning to optimize transmission over an unknown gilbert-elliott channel." In Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), IEEE (2012).

[20] Puterman, Martin L. "Markov Decision Processes: Discrete Stochastic Dynamic Programming (Wiley Series in Probability and Statistics)." (2005).

[21] Ross, Sheldon M. "Introduction to stochastic dynamic programming: Probability and mathematical." Academic Press, Inc., (1983).

[22] Gilbert, Edgar N. "Capacity of a Burst-Noise Channel." Bell system technical journal 39, no. 5: 1253-1265., (1960)

[23] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading." In INFOCOM, 2012 Proceedings IEEE, pp. 945-953. IEEE, (2012).

[24] Alexander Schrijver. Theory of linear and integer programming. John Wiley & Sons, (1998).

[25] Alpar Juttner, Balazs Szviatovski, Ildikó Mécs, and Zsolt Rajkó. "Lagrange relaxation based method for the QoS routing problem." In INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 2, pp. 859-868. IEEE, (2001).

[26] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. "The case for vm-based cloudlets in mobile computing." Pervasive Computing, IEEE 8, no. 4: 14-23 (2009).

[27] Paramvir Bahl, Richard Y. Han, Li Erran Li, and Mahadev Satyanarayanan. "Advancing the state of mobile cloud computing." In Proceedings of the third ACM workshop on Mobile cloud computing and services, pp. 21-28. ACM, (2012).

[28] Qi Zhang, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." Journal of internet services and applications 1, no. 1 : 7-18 (2010).