Making Offloading Decisions Resistant to Network Unavailability for Mobile Cloud Collaboration

Huijun Wu, Dijiang Huang
Arizona State University
{Huijun.Wu, Dijiang.Huang}@asu.edu

Samia Bouzefrane
Conservatoire national des arts et métiers
samia.bouzefrane@cnam.fr

***Abstract*—Offloading is one major type of collaborations between mobile devices and clouds to achieve less execution time and less energy consumption. Offloading decisions for mobile cloud collaboration involve many decision factors. One of important decision factors is the network unavailability that has not been well studied. This paper presents an offloading decision model that takes network unavailability into consideration. Network with some unavailability can be modeled as an alternating renewal process. Then, application execution time and energy consumption in both ideal network and network with some unavailability are analyzed. Based on the presented theoretical model, an application partition algorithm and a decision module are presented to produce an offloading decision that is resistant to network unavailability. Simulation results demonstrate good performance of proposed scheme, where the proposed partition algorithm is analyzed in different application and cloud scenarios.**

***Keywords*—Mobile Cloud Computing, Offloading, Reliability.**

I. Introduction

The collaboration of mobile devices and clouds enlarges the advantages of both mobile devices and clouds. The mobile devices can use the unlimited computation and storage resource of cloud, and meanwhile, the cloud gets more close to end user through the bridge of mobile devices. Mobile cloud application based on mobile cloud collaboration deploys its components into different places including local smart phone and virtual machines in cloud. The application initially starts all components on smart phone locally. When remote cloud resource becomes available, the mobile cloud application may offload computing-intensive or memory-consuming components to remote cloud to improve performance or save mobile device energy. One key issue in offloading process is how to make offloading decisions. Due to the complexity of mobile and cloud environments, offloading decision involves mobile side parameters such as CPU and memory, cloud side environment including virtual machine capability and performance, and the state of network in between, which altogether makes the decision making problem difficult to tackle. The network state is the most complicated and important part among three aspects considered.

Network status has great impact on mobile cloud collaboration. Due to mobility of mobile users, the network connecting mobile devices and clouds changes drastically. The network connection may be lost when a mobile device moves into some area that is not covered by wireless network. When the connection to cloud is lost, the original execution routine is interrupted and the application has to wait until the connection resumes [1]. The expected benefit of mobile cloud collaboration may not be obtained due to interruption of execution plan. The collaboration may even lead to negative impact to execution time or energy consumption in such scenario. Thus, we need to find offloading solution that is resistant to network unavailability to assure the less execution time and less energy consumption benefits.

To address the problem of offloading decision making in network with some unavailability, we present an offloading decision module that produces offloading decision that is resistant to network unavailability. The unstable network is modeled as an alternating renewal process that provides statistic information of connection duration. The proposed offloading decision module monitors network connection states and durations that are recorded in a history buffer. Then it calculates application partition that is aimed to give benefit in network with low availability. And it finally validates the partition and outputs the offloading decision. The offloading decision module contains two key components. One is for partitioning application components into smart phone and cloud sides based on unstable network assumption. The other is for validating the application partition by comparing the possible unavailability of the partition and the observed network unavailability. The proposed offloading decision module can be put into mobile cloud application to provide application offloading decisions with the consideration of network unavailability.

In summary, the contributions of presented research are highlighted as follow:

- *Mobile cloud collaboration model considering network unavailability:* The proposed scheme models the application execution time considering unstable network situation. The unstable network is modeled as an alternating renewal process. Then application execution time and energy consumption are analyzed in both ideal network and unstable network.
- *Application partition algorithm:* The proposed algorithm utilizes application information to find the application partition that can achieve both less execution time and less energy consumption when network availability is low. The algorithm works in a heuristic manner.
- *Bayesian decision approach:* The partition given by above partition algorithm is validated by comparing tolerated unavailability of given partition and observed network unavailability. Since the network states estimated based on history observation are not fully

COLLABORATECOM 2013, October 20-23, Austin, United States
Copyright © 2013 ICST
DOI 10.4108/icst.collaboratecom.2013.254106

trusted. The final offloading decision is made by a bayesian decision approach to mitigate the possible error.

- *Offloading decision module:* The proposed solution provides a module that can be plugged into mobile cloud applications to enable decision making in unstable network scenario. The offloading decision module consists of four parts: observation buffer, application partition component, network state predictor and partition validation component.

The paper is organized as follows. Section II introduces related work on mobile cloud offloading framework and offloading decision. Section III presents system and models in both ideal and unstable network situations. Section IV presents application partition algorithm and partition validation approach followed by offloading decision module. Section V presents simulation results and analysis. Section VI concludes the paper.

II. Related Work

In recent years, some research work on mobile cloud collaboration has been proposed. Cuckoo [2] and eyeDentify [3] proposed a static offloading framework based on Ibis [4] middle ware. The proposed framework is build on Android and requires a separate compile process to generate mobile device and cloud versions of the same application. This framework does not have offloading decision intelligence. Zhang *et al.* proposed a web service based elastic offloading framework [5]. This framework uses a cost model to make offloading decision. The author did not provide an effective method to obtain optimal offloading decision. ThinkAir [6] and MAUI [7] are energy saving frameworks. They profile the hardware components and make offloading to optimize energy usage of mobile devices. These two frameworks only consider the energy issue of offloading and ignore other aspects including network states. Clonecloud [8][9] and eXCloud [10] are virtual machine related frameworks. Clonecloud mirrors the mobile device application in the cloud. eXCloud migrates Java virtual machine stack to cloud. These two frameworks record the application state on mobile devices and resume the application from the stored state in cloud.

Besides mobile cloud collaboration frameworks, some research work focus on offloading decision making issues. Giurgiu *et al.* [11] models application as a dependency graph. 'All step' and 'K step' algorithms were proposed to solve application partition problem. Memory and transferred data size were taken into the consideration, but not network availability. Ou *et al.* [12] discussed system failure in offloading systems. They analyzed system execution time considering system failure and recovery. They did not consider how to make offloading decisions. Wolski *et al.* [13] considered offloading decision problem based on network bandwidth. They assumed that the network availability is not an issue. However, the network may even not be connecting in real mobile cloud computing scenario due to mobility or other reasons. Xian *et al.* [14] proposed an offloading method aimed at energy saving. The proposed method adjusts the time out threshold to decide whether to wait for remote cloud response or to resume execution locally. Gu *et al.* [15] proposed an offloading approach based on a fuzzy control model. They modeled mobile device situation as fuzzy parameters. Their approach considers optimization of wireless network bandwidth requirement and interaction delay. Sinha et al. [16] and Ou et al. [17] proposed offloading scenarios involving multiple sites. However, their approaches do not consider the network availability issue during the mobile cloud offloading process.

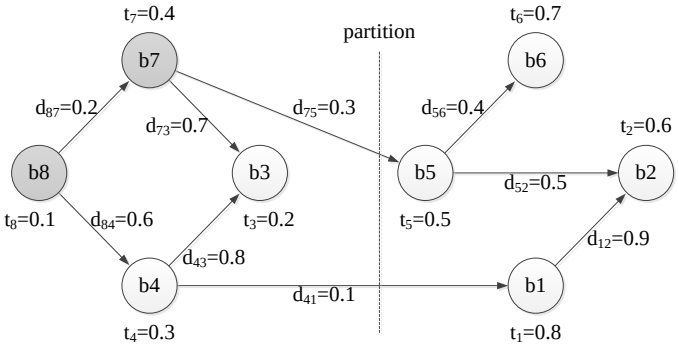


Fig. 1. Application graph example.

Although the above researches provided various offloading frameworks and decision models, network availability issues were ignored. We take network availability into consideration to better model real world scenarios.

III. System and Model

Mobile cloud consists of mobile devices and clouds. There is usually one to one mapping between mobile devices and virtual machines in cloud [18]. The mobile cloud application is constructed as a set of components or bundles. Some components can run on either mobile device or virtual machine, while the other components, such as user interface and sensors, have to run on mobile devices. The major offloading objectives are saving application execution time and energy consumption on mobile devices. Notations in following discussion are summarized in TABLE II in Appendix for convenience.

A. Application Model and Ideal Network Model

The application is presented as a directed acyclic graph $G = \{B, E\}$ where every vertex is a bundle and every edge is data exchange between bundles [11]. Each bundle has an attribute indicating whether it can be offloaded. The unmovable bundles are marked as *local* which means these bundles cannot be offloaded due to application requirements. Let m be the total count of bundles in the application, then the initial bundle set is $B = \{b_i \mid i \in [1, 2, \ldots, m]\}$ and the edge set is $E = \{e_{ij} \mid i, j \in [1, 2, \ldots, m]\}$ where e_{ij} represents directed data transfer from b_i to b_j. Let n be the count of movable bundles and $n \leq m$. A configuration c [11] is defined as a tuple of partitions from the initial bundle set, $< B_{phone}, B_{cloud} >$, where $B_{phone} = \{b_i \mid i \in [1, 2, \ldots, k]\}$ has k bundles and $B_{cloud} = \{b_i \mid i \in [1, 2, \ldots, s]\}$ has s bundles. And they satisfy $B_{phone} \cap B_{cloud} = \emptyset$ and $B_{phone} \cup B_{cloud} = B$. The bundles that are marked as *local* are initially put in the set B_{phone} and cannot be moved. An example is shown in Fig. 1 where unmovable bundles are marked as grey and dot line indicates configuration. The bundles on the left side are B_{phone} and the bundles on the right side are B_{cloud}.

1) Execution Time: For a given task, bundle b_i has an attribute t_i indicating its computation time on smart phone. And edge e_{ij} is associated with an attribute d_{ij} indicating the transferred date size from bundle i to bundle j. These value can be measured or estimated from collected application and device data. Total time of running task only on smart phone is

$$t_{phone} = \sum_{b_i \in B} t_i, \tag{1}$$

where data exchanges between bundles are not counted as they happen locally and cost little time compared to time of data exchange over network. For a particular configuration c, offloading rate p [12] is defined as the proportion of offloaded task to all task in terms of computation time. Then, task proportion is the same as time proportion due to the same processing capability on the same mobile device $p(c) = (\sum_{b_i \in B_{cloud}} t_i)/t_{phone}$, and $p(c)$ satisfies $0 \leq p(c) \leq 1$. Then, the computation time on smart phone is

$$\begin{aligned} t^{phone}_{computation}(c) &= \sum_{b_i \in B_{phone}} t_i = (\sum_{b_i \in B} - \sum_{b_i \in B_{cloud}}) t_i \\ &= (1 - p(c)) t_{phone}. \end{aligned} \tag{2}$$

Assume the cloud is q times faster than smart phone, thus the time consumption in clouds is q times less than the time spent on mobile devices [14], which is $t^{cloud}_i = t_i/q$. The computation time in cloud is

$$\begin{aligned} t^{cloud}_{computation}(c) &= \sum_{b_i \in B_{cloud}} t^{cloud}_i = \frac{1}{q} \sum_{b_i \in B_{cloud}} t_i \\ &= \frac{p(c)}{q} t_{phone}. \end{aligned} \tag{3}$$

Thus the total computation time is:

$$\begin{aligned} t_{computation}(c) &= t^{phone}_{computation}(c) + t^{cloud}_{computation}(c) \\ &= (1 - (1 - \frac{1}{q}) p(c)) t_{phone}. \end{aligned} \tag{4}$$

A typical offloading process works as follows. Initially, the application starts on smart phone and all components run locally. Then, the application may offload some components to remote virtual machines. These offloaded bundles run in cloud remotely. However they need to communicate with the bundles resident on smart phone. Thus they have to exchange data through network. Assume network bandwidth is w, then the network delay is the sum of delays in both data transfer directions:

$$t_{network}(c) = \sum_{\substack{b_i \in B_{phone} \\ b_j \in B_{cloud}}} \frac{d_{ij}}{w} + \sum_{\substack{b_i \in B_{cloud} \\ b_j \in B_{phone}}} \frac{d_{ij}}{w}. \tag{5}$$

In an ideal network environment, the total execution time for a given configuration c is the sum of computation time and network delay:

$$t(c) = t_{computation}(c) + t_{network}(c). \tag{6}$$

The offloading benefit of execution time comes from the trade of $t_{computation}$ and $t_{network}$. The computation part saves execution time because the cloud processing capability is powerful than mobile devices. However, the offloading has to pay network delay, which counteracts the computation time saving. For computation intensive applications whose computation time saving is much larger than network delay, the offloading benefit is obviously seen.

2) Energy Consumption: Two hardware modules of mobile devices are involved in the energy consumption estimation: CPU and radio frequency (RF) module. Other modules, like display, audio, GPS etc., are not considered because the components that interact with theses modules have to run on mobile devices locally. Both energy consumption on CPU and RF module can be further separated into dynamic part and static part [19]. When hardware module is in idle state, the energy consumption is corresponding to static part. When hardware module is in active state, more energy is consumed, which is corresponding to dynamic part. Assume the power of CPU in idle state is K^{sta}_{CPU} and the power of CPU in active state is $K^{sta}_{CPU} + K^{dyn}_{CPU}$. The energy consumption of CPU is:

$$P_{CPU}(c) = K^{sta}_{CPU} t(c) + K^{dyn}_{CPU} t^{phone}_{computation}(c). \tag{7}$$

Similarly, let K^{sta}_{RF} and K^{dyn}_{RF} be the power of RF module in idle and active state separately. The energy consumption on radio frequency module is:

$$P_{RF}(c) = K^{sta}_{RF} t(c) + K^{dyn}_{RF} t_{network}(c). \tag{8}$$

Thus, the total energy consumption is:

$$P(c) = P_{CPU}(c) + P_{RF}(c). \tag{9}$$

If offloading is not applied, only CPU consumes energy and its active period is the whole execution time. The total energy consumption of running tasks only on smart phone is:

$$P_{phone} = (K^{sta}_{CPU} + K^{dyn}_{CPU}) t_{phone}. \tag{10}$$

The offloading influences energy consumption of mobile devices in two aspects. First, the mobile device may save energy because the mobile device does not pay for the energy consumption corresponding to the tasks that are offloaded and completed in cloud. Second, the data exchange between application components are now fulfilled by networking instead of local procedure invocation, which leads to energy cost for sending and receiving packets. Similarly to time benefit of offloading, the computation intensive application may obtain obvious energy benefit when computation tasks are offloaded to save large CPU energy consumption and network energy consumption is small.

B. Model and Impact of Network Unavailability

The connection between mobile devices and clouds is usually not stable due to mobility of devices. When the mobile device moves out of wireless coverage, it loses connection to cloud. The mobile device continues to make attempts to reconnect to cloud when the network is unavailable. When it gets into coverage again, the connection resumes. As the mobile device moves, the connection state changes as *on*, *off*, *on*, *off* ..., which can be modeled as an alternating renewal process.

Fig. 2 shows how network availability changes along with time coordinate. Solid line represents network is available, while dash line represents network is unavailable. Two network

Fig. 2. Network unavailability model.

states alternate with each other. One *on* duration and one *off* duration form a cycle. The *on* state duration is denoted as ξ and the *off* state duration is denoted as η. $\{\xi_i, i = 1, 2, \ldots\}$ is independent and identically distributed (i.i.d.), and so is $\{\eta_i, i = 1, 2, \ldots\}$. And ξ_i and η_j are independent for any $i \neq j$, but ξ_i and η_i can be dependent [20]. The cycle duration is denoted as χ and $\chi_i = \xi_i + \eta_i$ where $i = 1, 2, \ldots$. The proportion of *on* duration in any individual cycle is a random variable denoted as $\rho = \xi/\chi$.

1) Execution Time: When the network is unavailable, the application has to wait because phone cannot send input to cloud and cannot retrieve output from cloud either. The application resume the execution after the network becomes available again. The total execution time is prolonged according to the proportion of ρ:

$$t'(c) = \frac{t(c)}{\rho}. \tag{11}$$

The offloading gives time benefit when $t'(c) < t_{phone}(c)$. In ideal network environment, $\rho = 1$ and $t'(c) = t(c)$. $t'(c)$ raises to infinity when ρ decreases from 1 to 0. At some point, the benefit disappears finally. We define this point as critical value of ρ for time benefit:

$$\rho'_{time}(c) = \frac{t(c)}{t_{phone}}. \tag{12}$$

The time benefit is:

$$\Delta t(c) = t_{phone}(c) - t'(c), \tag{13}$$

when $\rho > \rho'_{time}(c)$.

2) Energy Consumption: During time period $t'(c)$, the computation time $t'_{computation}(c)$ and network time $t'_{network}(c)$ are the same with $t_{computation}(c)$ and $t_{network}(c)$ in ideal network environment because computation and data transfer only work properly when network is available as ideal network. The CPU active time period is the same as that in ideal network model because the given task is the same. However, the CPU idle time period is the whole execution time that is different from that in ideal network model. Thus, the energy consumption for CPU is:

$$P'_{CPU}(c) = K^{sta}_{CPU} t'(c) + K^{dyn}_{CPU} t^{phone}_{computation}(c). \tag{14}$$

The RF module is active even when the network is unavailable because it continues scanning the available network to resume the connection. Thus, the active time period for RF module is $t'(c) - t_{computation}(c)$. The energy consumption for RF is:

$$P'_{RF}(c) = K^{sta}_{RF} t'(c) + K^{dyn}_{RF}(t'(c) - t_{computation}(c)). \tag{15}$$

Thus, the total energy consumption is:

$$P'(c) = P'_{CPU}(c) + P'_{RF}(c). \tag{16}$$

The offloading gives energy benefit if $P'(c) < P_{phone}$. As ρ decreases, both $P'_{CPU}(c)$ and $P'_{RF}(c)$ increase. Similarly, the critical value of ρ for energy is defined as the point where energy benefit disappears:

$$\begin{aligned}\rho'_{energy}(c) = (K^{sta}_{CPU} + K^{sta}_{RF} + K^{dyn}_{RF}) t(c)/(P_{phone} \\ - K^{dyn}_{CPU} t^{phone}_{computation}(c) + K^{dyn}_{RF} t_{computation}(c)).\end{aligned} \tag{17}$$

The offloading energy benefit is:

$$\Delta P(c) = P_{phone}(c) - P'(c), \tag{18}$$

when $\rho > \rho'_{energy}(c)$.

3) Problem Formulation: When network availability ρ is greater than the larger one of $\rho'_{time}(c)$ and $\rho'_{energy}(c)$, both time and energy benefit are obtained. We define the critical value of ρ is:

$$\rho'(c) = \max\{\rho'_{time}(c), \rho'_{energy}(c)\}. \tag{19}$$

The offloading problem with network unavailability consideration is to find the application partition c to minimize $\rho'(c)$:

$$\min\ \rho'(c), \tag{20}$$

while both time and energy benefit exist:

$$\rho > \rho'(c), \tag{21}$$

where ρ is the current network availability estimated based on observations. The c satisfying (21) may not exists when ρ is too low. In this situation, the application should not offload any components to cloud. The solution given by (20) is the best partition that tolerates the network unavailability, and it may give benefit when current network availability ρ goes worse.

IV. Algorithm and Solution

The bundle computation time t_i's form a vector $\mathbf{t}$ of m dimensions. The data size d_{ij}'s form a square matrix $\mathbf{D}_{m\times m}$. If there is no edge from b_i to b_j, then d_{ij} is set to 0.

The configuration c can be represented as a vector $\mathbf{x}$ of m dimensions where x_i indicates whether b_i should be offloaded. $x_i = 1$ means b_i should be offloaded to remote cloud, and x_i=0 means b_i should be kept on smart phone locally. For b_i that cannot be offloaded, x_i is set to 0 initially and does not change. Vector $\mathbf{x}$ has m elements in which n elements are variables and the others are 0s. For simplicity, all 0s are put at the end of $\mathbf{x}$.

Let $\mathbf{1}$ be a column vector whose elements are all 1s, then $t_{phone} = \mathbf{t}^T\mathbf{1}$. Offloading rate $p(c)$ is now $p(\mathbf{x}) = \mathbf{t}^T\mathbf{x}/\mathbf{t}^T\mathbf{1}$. And $t_{network}(c)$ is $t_{network}(\mathbf{x}) = ((\mathbf{1}-\mathbf{x})^T\mathbf{D}\mathbf{x} + \mathbf{x}^T\mathbf{D}(\mathbf{1}-\mathbf{x}))/w$. Thus $t(c)$ is finally function of $\mathbf{x}$, which is $t(\mathbf{x})$.

The objective of offloading decision is to find configuration $\mathbf{x}$ satisfying (20). This is a 0-1 Integer Programming (IP) problem.

A. Application Partition

The solution space for configuration $\mathbf{x}$ is 2^n, which means it costs a lot of time to search the optimal solution. To find proper $\mathbf{x}$ within acceptable time, we propose an Artificial Bee Colony (ABC) [21] based algorithm. The colony consists of three types of bees: employed bees, onlooker bees and scout bees. The bee that goes to food source visited by it before is

named employed bee. The bee that waits on the dance area for making a selection of food source is called onlooker bee. And the bee that carries random search for discovering new food source is named scout bee. The food source is a possible solution $\mathbf{x}$, and every bee can memorize one food source. It is assumed that there is only one employed bee for each food source. The memory of employed bees is considered as the consensus of the whole colony, and the food sources found by onlooker bees or scout bees merge into employed bees' memory in algorithm. Assume the number of employed bees is N and the number of onlooker bees is M ($M > N$). And let MCN be Maximum Cycle Number. The algorithm overview is shown in Algorithm 1.

Algorithm 1 Application partition algorithm overview.

```
Initialize employed bees.
cycle ← 1.
repeat
    Produce new solution for employed bees.
    Apply greedy selection process for employed bees.
    Determine probabilities, and assign M onlooker bees to
    N employed bees accordingly.
    Produce new solution for onlooker bees.
    Apply greedy selection process for onlooker bees.
    Determine abandon solution, if exists, and replace it
    with scout bee.
    Memorize the best solution so far.
    cycle ← cycle + 1.
until cycle = MCN
```

At the first step, the algorithm generates a random initial population X ($cycle = 0$) of N solutions where the population size is the same as number of employed bees. Based on this initial generation, the algorithm starts to evolve the generation in cycles. The evolution repeats until the cycle number reaches limit MCN. The algorithm outputs the best solution, denoted as $\mathbf{x}_{best}$, ever found in all cycles.

In the cycle, three types of bees work in sequence. The details of three type bees' actions are shown in Algorithm 2. Employed bees produce new solutions by two local search methods:

Flip employed bee randomly flips one element in the vector $\mathbf{x}$.

Swap employed bee randomly flips two elements of different values in the vector $\mathbf{x}$, which is equivalent to swapping two different elements in that vector.

Each employed bee evaluates the fitness of its original solution $\mathbf{x}$, new found $\mathbf{x}_{flip}$ and $\mathbf{x}_{swap}$ by (19). Then, each employed bee memorizes the best one of these three food sources and forgets the others.

Onlooker bees watch employed bee dancing, and plan the preferred food source. Onlooker bees record critical values of all food sources and calculate the probability for i^{th} food source as below:

$$p_i = \frac{1/\rho'(\mathbf{x}_i)}{\sum_{j=1}^{N}(1/\rho'(\mathbf{x}_j))}. \tag{22}$$

Intentionally, the lower food source's critical value is, the more likely onlooker bee would like to go. The onlooker bees chooses the food source $\mathbf{y}$ randomly according to its probability. Since $M > N$, several onlooker bees may follow the same employed bee and choose the same food source. Then each onlooker bee applies the same local search methods used by employed bees previously to explore new neighbour solutions, and picks the best one of the three. After all onlooker bees update their solution, each employed bee compares its solution with its followers' solutions, and picks the best one as its new solution.

Algorithm 2 Details of application partition algorithm.

```
Initialize employed bees randomly
  x_i: the i^th employed bee.
cycle ← 1.
repeat
  for each employed bee x_i do
    Apply Flip local search and find x_flip.
    Apply Swap local search and find x_swap.
    if ρ'(x_i) > min{ρ'(x_i), ρ'(x_flip), ρ'(x_swap)} then
      x_i ← arg min{ρ'(x_i), ρ'(x_flip), ρ'(x_swap)}.
    end if
  end for
  Determine probabilities (p_i) by (22).
  M_i ← p_i M: number of onlooker bees sent to
    the i^th food source.
  y_ij ← x_i (j = 1, 2, ..., M_i): the j^th onlooker bee
    of the i^th food source.
  for each onlooker bee y_ij do
    Apply Flip local search and find y_flip.
    Apply Swap local search and find y_swap.
    if ρ'(y_ij) > min{ρ'(y_ij), ρ'(y_flip), ρ'(y_swap)} then
      y_ij ← arg min{ρ'(y_ij), ρ'(y_flip), ρ'(y_swap)}.
    end if
  end for
  for each employed bee x_i do
    if ρ'(x_i) > min_{j=1,2,...,M_i} {ρ'(y_ij)} then
      x_i ← arg min_{j=1,2,...,M_i} {ρ'(y_ij)}.
    end if
  end for
  Generate scout bee z randomly.
  if max_{i=1,2,...,N} {ρ'(x_i)} > ρ'(z) then
    arg max_{i=1,2,...,N} {ρ'(x_i)} ← z.
  end if
  if ρ'(x_best) > min_{i=1,2,...,N} {ρ'(x_i)} then
    x_best ← arg min_{i=1,2,...,N} {ρ'(x_i)}.
  end if
  cycle ← cycle + 1.
until cycle = MCN
```

In our algorithm, only one scout bee is used. This scout bee randomly generates vector $\mathbf{z}$ and compares $\mathbf{z}$ to the worst solution of employed bees. If this random generated $\mathbf{z}$ is better than the worst solution of employed bees, the corresponding employed bee memorizes this new solution and forgets the old one.

B. Bayesian Decision

To complete the last step of decision, ρ has to be estimated from the observations $\{\xi_i, i = 1, 2, \ldots, n\}$ and $\{\eta_i, i = 1, 2, \ldots, n\}$, and then compared with $\rho'(c)$ according to (21). We assume there is a module named *Predictor* that fulfills this last step. In our implementation, the *Predictor* calculates the average value of ρ and outputs the comparison of this average value and $\rho'(c)$. However, this implementation is only a simple one, and it can be replaced with other advanced implementation. Thus, the *Predictor* module can be treated as a black box [13].

Although the observations are known, the distribution of ρ is unknown in most scenarios. Thus, to estimate ρ analytically is not practical. Besides, the network is always dynamic, which is not fully understood yet. Therefore, to predict future network state based on historical observation is usually done empirically. From this perspective, the comparison result from black box *Predictor* cannot be fully trusted. Thus, we use another module named *Bayesian Decision* to mitigate the errors of *Predictor* and to make the final decision for (21).

For simplicity, let T be $\rho'(c)$. And we assume $\omega_{offload}$ denotes the comparison result $\rho > \rho'(c)$, and ω_{local} denotes $\rho \leq \rho'(c)$. The Bayesian risk of decision to offload is the expected cost as a function of the posterior distribution associated with offloading:

$$\begin{aligned} R_{offload} &= \int_0^T \mathrm{C}_{offload}(\rho) f(\rho|\omega_{offload}) \mathrm{d}\rho \\ &= \mathrm{C}_{offload}(\rho) \mid_{\rho = \int_0^T f(\rho|\omega_{offload})\mathrm{d}\rho}, \end{aligned} \tag{23}$$

where loss function $\mathrm{C}_{offload}(\rho)$ can be either time expense (11) or energy expense (16) according to user preference. And $f(\rho|\omega_{offload})$ is the probability of observing a network availability value ρ given the *Predictor*'s output $\omega_{offload}$. When ρ is greater than T, offloading does not pay penalty, thus the integral range is $[0, T]$. Similarly, the risk of decision to execute locally is:

$$\begin{aligned} R_{local} &= \int_T^1 \mathrm{C}_{local} f(\rho|\omega_{local}) \mathrm{d}\rho \\ &= \mathrm{C}_{local} \int_T^1 f(\rho|\omega_{local}) \mathrm{d}\rho, \end{aligned} \tag{24}$$

where loss function C_{local} is constant value calculated by either (1) or (10) according to the same user preference used for (23). We may pay penalty in both offloading or local execution scenarios, which is the risk behind the above two equations. When offloading is applied, the execution time or energy consumption may exceed the cost of local execution according to (11) and (16). Similarly, if local execution is chosen, the benefit may be lost when (21) is satisfied.

To calculate (23), the posterior Probability Distribution Function (PDF) $f(\rho|\omega_{offload})$ can be calculated by Bayes theorem:

$$f(\rho|\omega_{offload}) = \frac{f(\omega_{offload}|\rho) f(\rho)}{f(\omega_{offload})}, \tag{25}$$

where $f(\rho)$ is the prior PDF of availability ρ, $f(\omega_{offload}|\rho)$ is the conditional PDF of event $\omega_{offload}$ given a availability ρ. $f(\rho|\omega_{local})$ is calculated in the same way for (24).

We maintain a histogram for $f(\rho)$. The value range $[0, 1]$ is divided into many slots, each of which is indexed by a ρ value, and each of which is corresponding to a bin in histogram. When a particular ρ value is observed, it falls into one bin. The bins accumulate the observation count. To calculate $f(\omega_{offload}|\rho)$ and $f(\omega_{local}|\rho)$, We maintain two counters for each bin in the histogram. One counter is for counting event $\omega_{offload}$ and the other is for counting ω_{local}. The events $\omega_{offload}$ and ω_{local} are counted according to their corresponding bin indexed by availability ρ. There are another two global counters that count the events $\omega_{offload}$ and ω_{local} no matter what availability value ρ is associated. These two counters are used to calculate $f(\omega_{offload})$ and $f(\omega_{local})$.

For example, assume the histogram has ten bins, which means the range $[0, 1]$ is divided into ten slots. The first bin is corresponding to range $[0, 0.1]$; the second bin is corresponding to range $[0.1, 0.2]$, and so forth. When we observe an availability value $\rho = 0.47$, we add 1 to the fifth bin $[0.4, 0.5]$. Assume the corresponding event of this observation is ω_{local}, we add 1 to the counter of ω_{local} associated with the fifth bin, and add 1 to the global counter of ω_{local} as well.

C. Offloading Decision Module

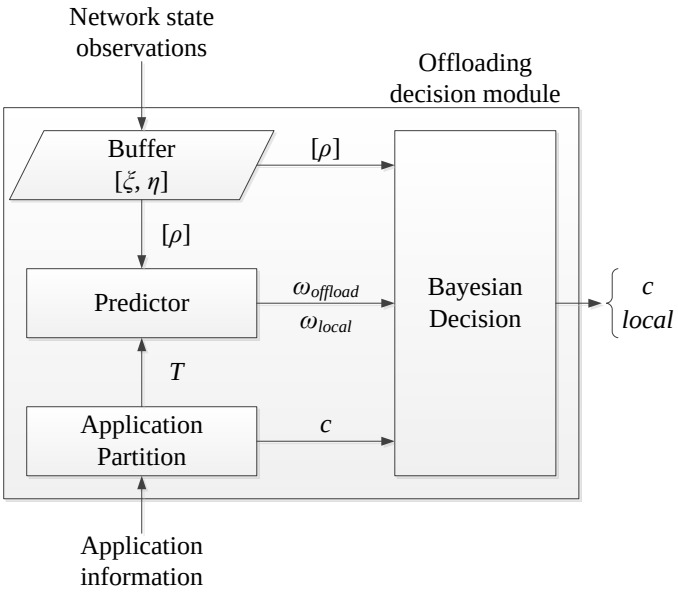
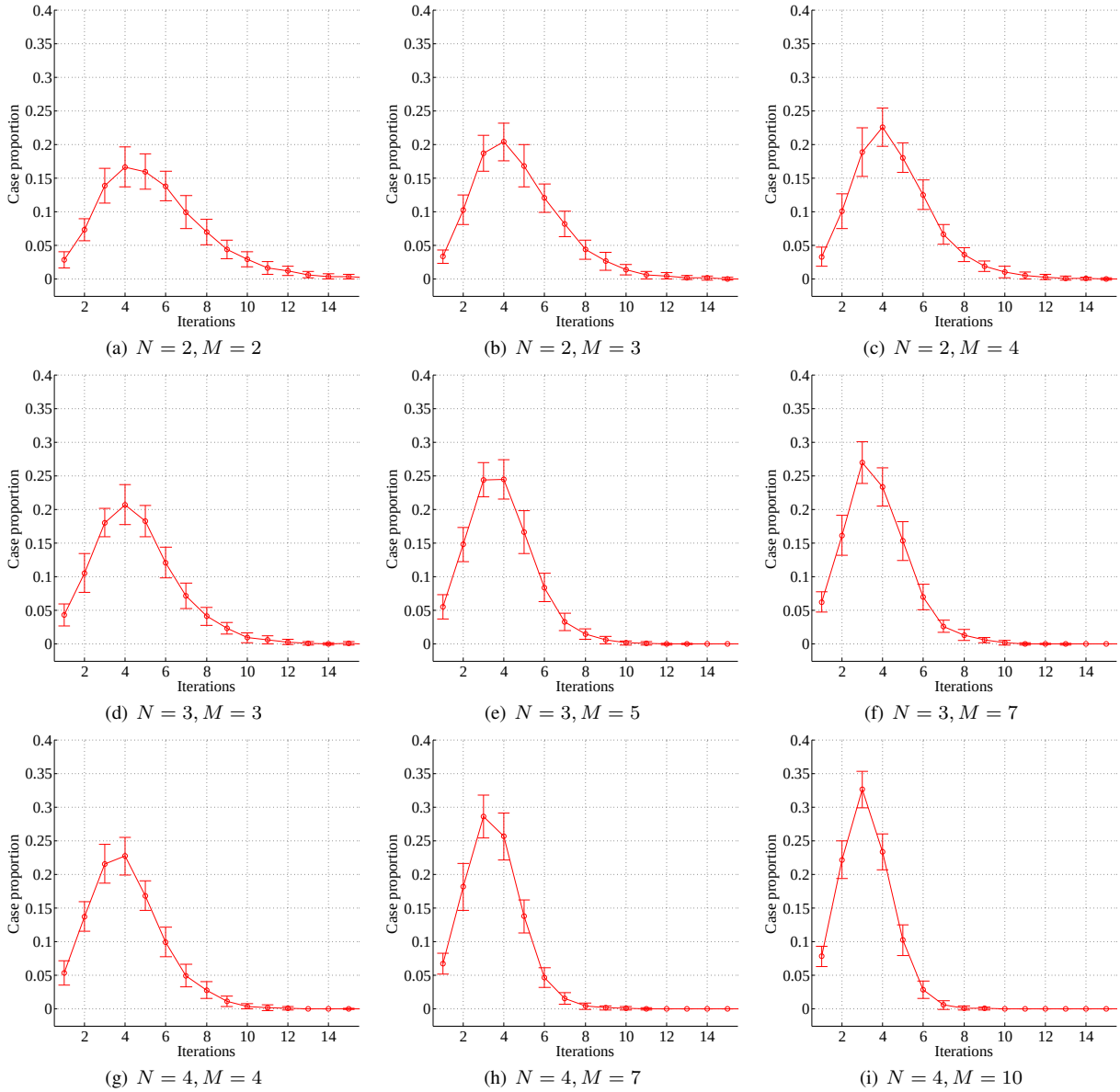


Fig. 3. Offloading decision module.

Based on the above discussions, the offloading decision module is depicted in Fig. 3. The offloading decision module maintains a buffer that stores network states duration sequences. The buffer is constructed as a table that has two columns ξ and η. Every row is an observation of cycle of alternating renewal process. The rows are sorted with the latest observation on the top and oldest observation at bottom. When the table is full, the oldest tuples are removed from the bottom. When application information is provided, *Application Partition* executes the algorithm in section IV-A and outputs threshold T and configuration c. *Predictor* reads buffer and T, then gives availability prediction. Then *Bayesian Decision* finally makes the decision according to section IV-B. The final output of the offloading decision module is either a configuration or a flag indicating local execution.

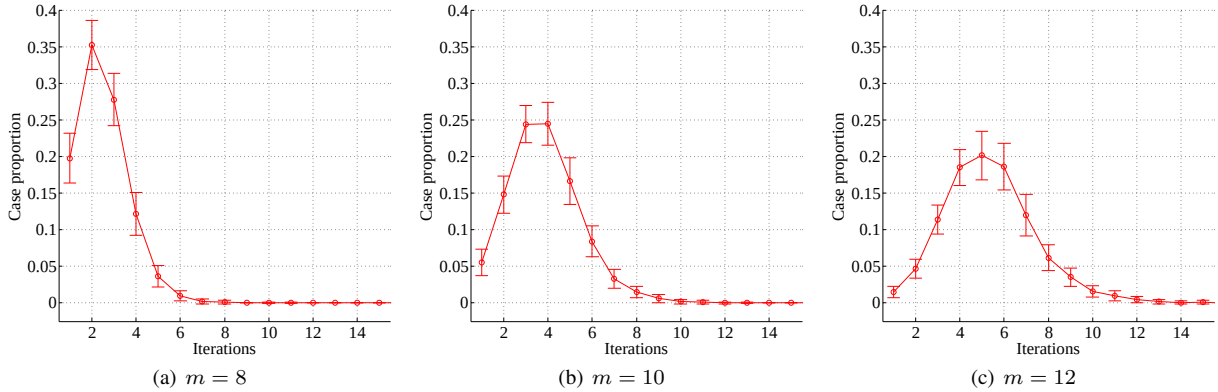


(a) $N = 2, M = 2$ (b) $N = 2, M = 3$ (c) $N = 2, M = 4$

(d) $N = 3, M = 3$ (e) $N = 3, M = 5$ (f) $N = 3, M = 7$

(g) $N = 4, M = 4$ (h) $N = 4, M = 7$ (i) $N = 4, M = 10$

Fig. 4. Partition performance of difference bee colonies.



(a) $m = 8$ (b) $m = 10$ (c) $m = 12$

Fig. 5. Partition performance of different component numbers.

V. Simulation and Analysis

In this section, we evaluate ABC based partition algorithm including algorithm tuning and impact of different mobile application and cloud. We evaluate our model and algorithms in MATLAB.

We generate two hundred random application graphs as base evaluation data set. The default parameter settings are shown in TABLE I. We use a set of typical energy parameters K for a phone according to [19]. The cloud-phone processing ability ratio q varies in large range from previous work [13][14]. We pick a medium value from the possible range as default value and evaluate its impact to algorithm in section V-C. We evaluate the algorithm in three aspects: bee colony, different applications and cloud-phone relation.

TABLE I. Default Parameter Setting

Parameter		Default value
Application	m	10
	n	8
Cloud [13][14]	q	30
Phone [19]	K_{CPU}^{sta}	2.5
	K_{CPU}^{dyn}	5
	K_{RF}^{sta}	1.25
	K_{RF}^{dyn}	1.25
Algorithm	N	3
	M	5

A. Bee Colony and Algorithm Tuning

This experiment is based on two hundred random application graphs (**t** and **D**). These application graphs are randomly generated, and at least one configuration of each graph is guaranteed to obtain both time and energy benefit in ideal network environment. We evaluate the proposed algorithm performance of difference bee colony size. The results are shown in Fig. 4. The x-axis represents how many iterations that the algorithm needs to reach the $\mathbf{x}_{best}$, and y-axis represents how many cases reach the solution of corresponding iterations. From the figure, we may draw following guides for algorithm tuning:

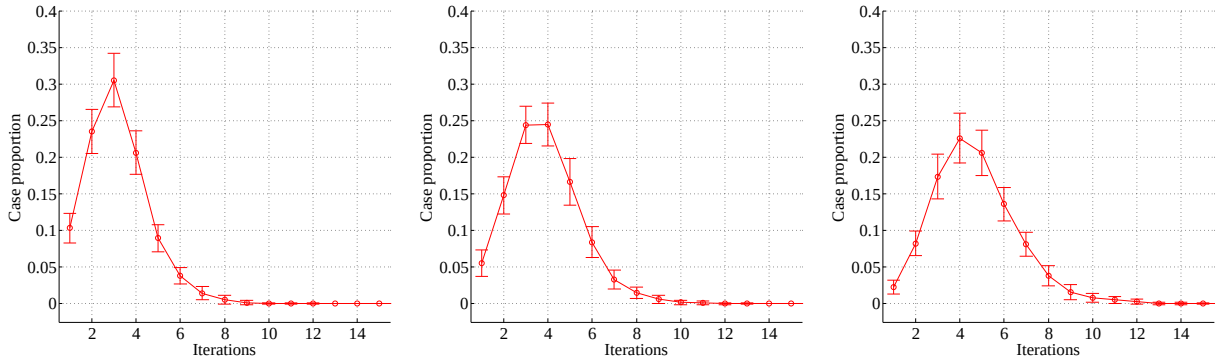
- Increasing onlooker bee number, algorithm shows the better convergent rate. For the same employed bee number (N), the more onlooker bees there are, the less iterations are required to reach the optimal solution obviously in all three situations ($N = 2, 3, 4$).
- Increasing employed bee number improves convergent rate, but the improvement is not obvious compared to increasing onlooker bee number. For the same onlooker bees of $M = 3$ (Fig. 4(b), Fig. 4(d)), $M = 4$ (Fig. 4(c), Fig. 4(g)) and $M = 7$ (Fig. 4(f), Fig. 4(h)), the cases that have more employed bees have slightly performance improvement, which is not as obvious as the improvement given by increasing onlooker bees. For $M = 4$ figures, more than 0.05 cases reach the optimal solution at iteration number 7 in Fig. 4(c) while there are only 0.05 cases reach solution at iteration number 7, which means more cases reach solution in less than 7 iterations, in Fig. 4(g). Similar phenomena is found for $M = 3$ and $M = 7$ figures.
- For the same total bee number of employed bees and onlooker bees, the algorithm prefers more onlooker bees slightly. For the same total bees of $N + M = 6$ (Fig. 4(c), Fig. 4(d)) and $N + M = 8$ (Fig. 4(e), Fig. 4(g)), we can see that the overall performance are almost the same. But the iterations to get optimal solutions in the cases that have more onlooker bees are slightly concentrated on some iteration numbers. For $N+M = 6$, the iterations in Fig. 4(c) are concentrated demonstrated by higher summit at iteration 4, while it is diversely distributed from 1 to 8 in Fig. 4(d). Similar situation occurs for $N + M = 8$.

B. Application Impact

To evaluate the algorithm performance for difference applications, three experiments are done for component number, unmovable component proportion and computation-communication ratio separately. The experiment result for difference component number is shown in Fig. 5. For the same bee colony, the iterations to find the solution increases along with the component number. For the large applications that have many components, the algorithm may use large bee colony to assure small iterations.

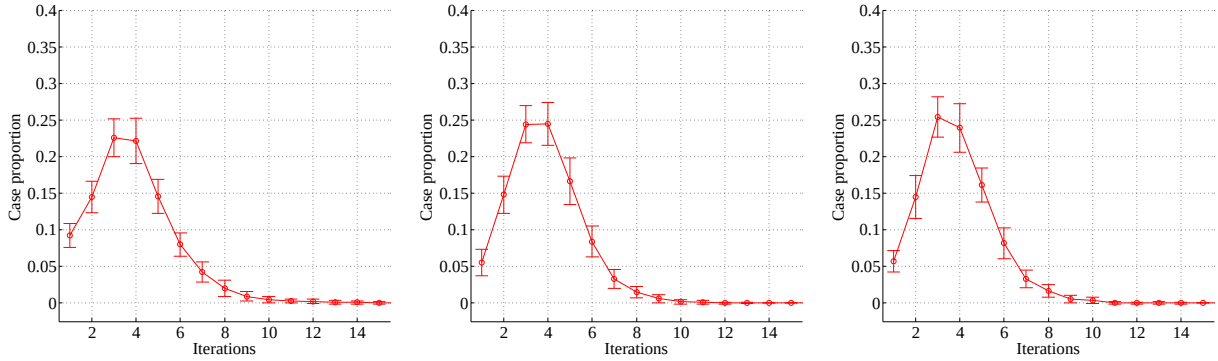
We evaluate the impact of unmoveable component proportion of application in the second experiment and the result is shown in Fig. 6. From the figure, we find that more iterations are required to get the solution when the movable component number increases. The trend is very like that in Fig. 5, which implies that the unmoveable component number does not play much role in the algorithm performance. This is because the algorithm always consider the movable components and ignore the unmovable component when generating new solutions in each cycle. The total component size increase in Fig. 5 leads to the increase of movable component number, which is like what this experiment does in Fig. 6. Besides, we also found in this experiment that higher movable proportion results in the robust solution that can work under low network availability $\rho'(c)$ situations. This is because the high movable proportion provides more candidate partition options so that more robust solution may be achieved.

We generate another two sets of application graph data of different computation load. The data set used in previous experiments are used as reference data set. Then we adjust the computation task to half and double of the reference data set in the application graph generation process. The network task remains the same, thus the computation-communication ratio is adjusted to half and double in new data sets. The experiment results for these three data set are shown in Fig. 7. From the figure, we can see that the iterations, distributed at 2,3,4,5 and 6, are almost the same, which implies the computation proportion of the given task does not influence the algorithm performance. This is because the computation proportion in the task influence the time benefit and energy benefit in the same direction. In the experiment, we also find that the computation proportion impacts the $\rho'(c)$, because the more computation proportion leads to more offloading benefit and the possible solution is more resistent to network unavailability.



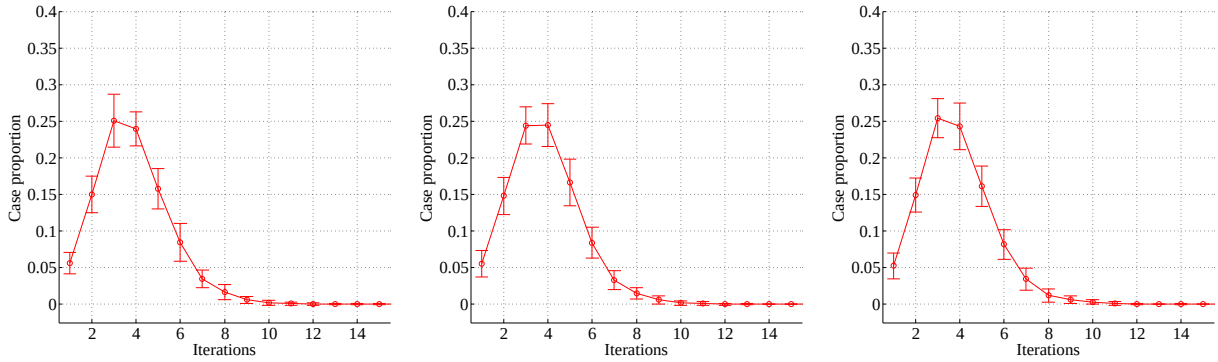
(a) $n = 7$ (b) $n = 8$ (c) $n = 9$

Fig. 6. Partition performance of different unmovable component proportions.



(a) half computation task (b) reference computation task (c) double computation task

Fig. 7. Partition performance of different computation tasks.



(a) $q = 5$ (b) $q = 10$ (c) $q = 20$

Fig. 8. Partition performance of different cloud speedup ratios.

C. Cloud Impact

We evaluate the algorithm performance under different cloud speedup ratios shown in Fig. 8. The figure shows that the iteration number does not depend on the cloud processing capability. The cloud processing capability influences the execution time considered in algorithm, which is similar to the computation proportion impact. And similarly, the higher cloud processing capability results in more robust partition configuration.

VI. Conclusion

In this paper, we proposed an offloading decision model for mobile cloud application. The proposed solution modeled unstable network as an alternating renewal process. The execution time and energy consumption are analyzed in this unstable network scenario. The offloading problem is formulated as an optimization problem to find an application partition configuration that can provide offloading benefit when low network availability is low. A bee colony based algorithm

was proposed to calculate the application partition resistant to network unavailability. And a bayesian decision approach was proposed to validate the partition and output the final offloading decision. The simulation results demonstrated good performance of proposed solution.

ACKNOWLEDGMENT

The authors would like to thank ONR Young Investigator Program (YIP) award and NSF CPS #1239396 grant to support the research on the MobiCloud project. The authors would also like to thank the anonymous reviewers for their constructive comments and suggestions to improve the research quality of this work.

REFERENCES

[1] J. Flinn, "Cyber foraging: Bridging mobile and cloud computing," *Synthesis Lectures on Mobile and Pervasive Computing*, vol. 7, no. 2, pp. 1–103, 2012.

[2] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*. Springer, 2012, pp. 59–79.

[3] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen, and H. Bal, "eyedentify: Multimedia cyber foraging from a smartphone," in *IEEE International Symposium on Multimedia (ISM)*. IEEE, 2009, pp. 392–399.

[4] N. Palmer, R. Kemp, T. Kielmann, and H. Bal, "Ibis for mobility: solving challenges of mobile computing using grid techniques," in *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*. ACM, 2009, p. 17.

[5] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Unleashing the power of mobile cloud computing using thinkair," 2011.

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[8] B.-G. Chun, S. Ihm, P. Maniatis, and M. Naik, "Clonecloud: boosting mobile device applications through cloud clone execution," 2010.

[9] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution." in *HotOS*, vol. 9, 2009, pp. 8–11.

[10] R. K. Ma, K. T. Lam, and C.-L. Wang, "excloud: Transparent runtime support for scaling mobile applications in cloud," in *International Conference on Cloud and Service Computing (CSC)*. IEEE, 2011, pp. 103–110.

[11] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," *Middleware*, pp. 83–102, 2009.

[12] S. Ou, Y. Wu, K. Yang, and B. Zhou, "Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments," in *IEEE International Conference on Communications (ICC)*. IEEE, 2008, pp. 1856–1860.

[13] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, 2008, pp. 1–8.

[14] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *International Conference on Parallel and Distributed Systems*, vol. 2. IEEE, 2007, pp. 1–8.

[15] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2003, pp. 107–114.

[16] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2011, pp. 184–194.

[17] S. Ou, K. Yang, and A. Liotta, "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems," in *Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2006.

[18] D. Huang, "Mobile cloud computing," *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter*, vol. 6, no. 10, pp. 27–31, 2011.

[19] Y. Wang, X. Lin, and M. Pedram, "A nested two stage game-based optimization framework in mobile cloud computing system." in *SOSE*, 2013, pp. 494–502.

[20] T. Pham-Gia and N. Turkkan, "System availability in a gamma alternating renewal process," *Naval Research Logistics (NRL)*, vol. 46, no. 7, pp. 822–844, 1999.

[21] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied Mathematics and Computation*, vol. 214, no. 1, pp. 108–132, 2009.

APPENDIX

TABLE II. NOTATIONS.

Terms	Meaning
B	bundle set
b_i	a bundle
e_{ij}	service dependency from b_i to b_j
B_{phone}, B_{cloud}	bundle set for smart phone and cloud separately
t_i	execution time of b_i
c	partition configuration
t_{phone}	application execution time in non-offloading mode
$p(c)$	offloading rate
q	how faster cloud is than smart phone
d_{ij}	size of data transferred over e_{ij}
w	bandwidth
$t(c)$	total execution time
$t'(c)$	total execution time in unstable network
ξ, η	*on* state duration, *off* state duration
χ	cycle duration, $\chi = \xi + \eta$
ρ	network availability, $\rho = \xi/\chi$
$P_{CPU}(c)$, $P_{RF}(c)$	energy consumption for CPU and RF separately
$P'_{CPU}(c)$, $P'_{RF}(c)$	energy consumption in unstable network
$P(c)$	total energy consumption on a mobile device
$P'(c)$	total energy consumption in unstable network
P_{phone}	energy consumption in non-offloading mode
K^{sta}_{CPU}, K^{dyn}_{CPU} K^{sta}_{RF}, K^{dyn}_{RF}	power of CPU and RF on smart phone
$\rho'_{time}(c)$, $\rho'_{energy}(c)$	critical value for time and energy benefit
$\rho'(c)$	circuital value for both time and energy benefit
$\Delta t(c)$, $\Delta P(c)$	time benefit, energy benefit