International Journal of Computer Applications (0975 - 8887)
Volume 132 - No.3, December 2015

Application Method-based Efficient Offloading Scheme in Mobile Cloud Computing

Ahmed A.A. Gad-ElRab
Department of Mathematics
Faculty of Science
Al-Azhar University, Cairo, Egypt

T.A.A. Alzohairy
Department of Mathematics
Faculty of Science
Al-Azhar University, Cairo, Egypt

Farouk A. Emara
Department of Mathematics
Faculty of Science
Al-Azhar University, Cairo, Egypt

ABSTRACT

Mobile Cloud Computing is a new paradigm that transfers the data storage and the data processing from a mobile device to a power full cloud server which has a big storage. Mobile cloud applications use offloading schemes to move the computing power and data storage away from mobile phones into this cloud server. However,for a code compilation, offloading might consume more energy than the local processing of data when the size of code is small. So, a new offloading schemes are needed to be adaptive with the code size of an application or a service. This paper introduces a new method-based offloading scheme for mobile application. The proposed scheme divides each mobile application into presentation layer, logical layer and data access layer. Also, it considers each service or process in each layer as a set of methods. The methods of presentation layer resides on the mobile device, the methods of data layer is fully deployed on the cloud to minimize the data access, and the methods of logic layer is distributed between the cloud and mobile device by using formulated cost model which takes into account energy, memory, time, and data transfer delay costs. The conducted simulation results show that the offloading performance of the proposed scheme is much better than local processing scheme.

Keywords

Application partitioning, Battery Consumption, Mobile cloud computing,Offloading

1. INTRODUCTION

Mobile Cloud Computing refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones to the cloud which brings applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers[1]. Computation offloading offloads intensive methods of mobile applications to run remotely on rich resource such as cloud. In case of code compilation, offloading might consume more energy than that of the local processing when the size of codes is small. So, offloading is not always the effective way to save energy of a mobile device. For example, when the size of altered codes after compilation is 500 KB, offloading consumes about 5% of a device's battery for its communication while the local processing consumes about 10% of the battery for its computation. In this case, the offloading can save the battery up to 50%. However, when the size of altered codes is 250 KB, the efficiency reduces to 30%. Also computation offloading may require a large amount of data to be transferred on runtime, then higher latencies may occur.

In recent years, a lot of studies have been appeared to support remote execution for mobile applications on the cloud to increase the performance and reduce energy consumption[2, 3]. Generally, there are two main approaches which are introduced to perform remote execution. The first approach is to use full process or full VM (Virtual Machine) migration as in CloneCloud[4]. The full process or VM can be migrated to the rich infrastructure to execute remotely. While the second approach is only offloading intensive methods or services of applications to execute remotely[5, 6]. This approach leads to large energy saving because it is fine grained applications. This means that it can remote only the sub-parts that benefit from remote execution[5, 7].
The main problems of the first approach are: (1) access to native resources that are not virtualized already and are not available on clone, (2) may be offloading task not correct if it consumes energy more than running local, and (3) computation offloading may be requires a large amount of data to be transferred on runtime, then higher latencies may occur. While the main problems of the second approach are:(1) the cost model has focused so far only on the client side(mobile device) and has assumed the server's resources(cloud) to be infinite[6]. (2) CPU consumption and energy consumption are not including into the optimization problem[6]. (3) offloading software block (bundle) needs high bandwidth [6]. (4) solving the optimization problem for all methods in application that are marked as remote method this consume energy[5]. (5) memory cost is not including into the optimization problem [5]. (6) if the programmer forgets to mark methods (for remote execution), MAUI will not be able to offload those methods [5]. (7) MAUI does not consider the execution time in its optimization cost however it can predict the execution of a method [5].
In this paper to solve these problems method-based offloading scheme for mobile application is proposed. The proposed scheme divides each mobile application into three layers:(1) presentation layer, (2) logical layer and (3) data access layer. Also, it considers each service or process in each layer as a set of methods. The methods of presentation layer resides on the mobile device, the methods of data layer is fully deployed on the cloud to minimize the data access, and the methods of logical layer is distributed

between the cloud and mobile device by using formulated cost model which takes into account energy, memory, time, and data transfer delay costs.
The rest of this paper is organized as follows: the related work will be introduced in Section 2. Section 3 describes offloading problem formulation in MCC. Section 4 explains the proposed scheme. Section 5 introduces the simulation and qualitative evaluation of the proposed scheme and Section 6 concludes the paper.

2. RELATED WORK

In recent years, a lot of studies have been appeared to support remote execution for mobile applications on the cloud[4, 5, 6]. In the rest of this section, these related work will be introduced in details.

2.1 CloneCloud

CloneCloud is introduced by B. Chun[4] in 2011. The concept of clonecloud is based on creating virtual Smartphone on the cloud. The virtual Smartphone on the cloud has more hardware, software, network and energy which provides more suitable environment to process complicated tasks. The partitioning mechanism in CloneCloud is divide the application into software blocks based on energy consumption intensive or computing. Some of this block will be running on the Smartphone and other will be running on the clone (virtual Smartphone on the cloud). Once the virtual Smartphone is available, then the Some computing or energy-intensive blocks will be offloaded to cloud for processing. Once those execution blocks have been completed, the output will be passed from virtual Smartphone on the cloud to the Smartphone.
The main disadvantages of CloneCloud approaches are: (1) access to native resources that are not virtualized already and are not available on clone. (2) may be offloading task not correct if it consumes energy more than running local. (3) computation offloading may require a large amount of data to be transferred on runtime, then higher latencies may occur.

2.2 Giurgiu et al. Model

Giurgiu et al.[6] proposed a model that focuses on offloading intensive parts of applications to execute remotely on the cloud/server. The main objective of this model is to optimize latency, data transfer delay and cost. The core method of this model using R-OSGi[8] and AlfredO[9] frameworks for the management and deployment of applications. R-OSGi is an enhanced version of OSGi that supports multiple VMs residing on distributed servers, whereas the primary objective of OSGi is to assist with the decomposition and coupling of applications into modules, called bundles. The proposed model divides each mobile application into presentation layer, logical layer and data access layer. AlfredO distributes the bundles of layers between the Smartphone and server . The bundles of presentation layer reside on the Smartphone while the bundles of logical layer are distributed between the server and the Smartphone. Moreover, the bundles of data layer is fully deployed on the server to minimize the data access delay.
The main disadvantages of Giurgiu et al. are:(1) the cost model has focused so far only on the client side(mobile device) and has assumed the server's resources(cloud) to be infinite.(2) CPU consumption and energy consumption are not including into the optimization problem and (3) offloading software block (bundle) needs high bandwidth

2.3 MAUI Model

MAUI[5]Provides fine-grained application code offloading with minimum programmer intervention. The main objective of this model is to minimize energy consumption of mobile devices, which is the foremost challenge of the mobile industry. Therefore, MAUI offloads all the resource intensive methods to the nearby infrastructure or cloud, provided the offloading is beneficial in terms of energy. MAUI uses a profiler (optimization engine) that analyzes energy consumption involved in the local and remote execution of the code. Moreover, MAUI profiles offload methods and use history based approach to predict the execution time of a particular code. Therefore, if the remote execution is beneficial in terms of energy, then the code is offloaded to the nearby infrastructure. In MAUI, the application partitioning is dynamic and the offloading is done on the basis of methods instead of complete application modules to minimize the offloading delay. However, MAUI creates two versions of smartphone application, for local and remote execution using Microsoft .NET Common Language Runtime (CLR). In MAUI, the mobile device consists of three main components, i.e., solver interface, profiler and client proxy. The solver interface provides interaction with the solver (decision engine) and facilitates the offloading decision making. The profiler collects information regarding the application energy consumption and data transfer requirements. The client proxy deals with the method offloading and data transfer. Similarly, the server side consists of profiler, server proxy, solver and controller. However, the working of a profiler and server proxy is similar to the smartphone. The solver is the main decision engine of the MAUI that holds the call graph of the applications and the scheduled methods. Lastly, the controller is responsible for the authentication and resource allocation for incoming requests. However, single method offloading is less beneficial compared to combined methods (multiple methods) offloading. Another weakness of MAUI is that if the programmer forgets to mark methods (for remote execution), MAUI will not be able to offload those methods. Also, MAUI does not consider the execution time in its optimization cost however it can predict the execution of a method. Nevertheless, the MAUI profilers consume processing power, memory and energy, which is an overhead on the smartphones.
To solve the before mentioned problems of current approaches, a new application method-based offloading scheme is proposed. The proposed model divides an application into three layers: (1) presentation layer which contains user interface and resides on the smartphone, (2) logical layer which contains computation methods and is distributed between the cloud and the smartphone according to determined optimal cost which takes into account memory constrain, and (3) data layer which contains data and data access method and is fully deployed on the cloud to minimize the data access over the data layer. Instead of offloading a whole service or a whole application to the cloud, the proposed model works with methods of each service by adaptively offloads some of the logical layer methods of a service based on a determined cost model.

3. OFFLOADING PROBLEM IN MCC

3.1 Problem Description

In MCC, due to the resource-limited devices (i.e., Mobile devices) that contact with the resourceful machines (i.e., cloud servers), migrating the large computations and complex processing of certain services or methods from these devices to the servers is required to minimize energy consumption of mobile devices. This migrating process is called Offloading process. The decision of computation offloading is an extremely complex process and is affected by the

nature of the application. For instance, an application that requires local hardware resources (GPS, camera, and sensors) may not be able to execute in the cloud unless the application is partitioned into components, and local-resource independent components are moved to the cloud. However, offloading is not always the effective way to save energy. Therefore, the offloading problem is how services or methods can be offloaded such that the mobile devices can save their energy with keeping a high service performance and minimum time delay.
In the reset of this section, the problem assumptions and models will be described. Then, the offloading problem will be formulated.

3.2 Assumptions and Models

The MCC model consists of a mobile node MN and a cloud server node CS. MN can communicate with CS by using advanced wireless technology to exchange data, applications, and services. In addition, the set of assumptions that must be met in this MCC model are (1) there are developers can apply the Model-view-controller (MVC) [10] design pattern explicitly and rigorously to isolate the application logical layer from the user interface and data layer of any mobile application,(2) any method that interacts with a user or needs to access device hardware will belong to a user interface, (3) the energy consumption of each hardware component of MN such as LCD, CPU and Wi-Fi can be measured separately by using a measurement application model for the energy consumption on a mobile device (e.g., Android phone) on the fly[11], and (4) any mobile application consists of a set of methods and each method consists of a set of instructions that can be determined at run time. The set of methods in a logical layer of a certain service which can be offloaded is denoted as SM =$\{m_i, 1 \leq i \leq n\}$. Each $m_i \in$ SM has several metadata properties as memory cost, mem_i and code size, $code_i$.The number of instructions in a code size cod_i of a method $i$is denoted as I. The data size is needed for a method i to be sent or received are denoted by $send_{i,s}$ and $recv_{i,r}$, respectively. In addition, the speed of executing any instruction by a mobile node is denoted by $Speed_{MN}$. Finally, there are n methods that can be offloaded for an application or a service.

3.3 Problem Formulation

To formulate the offloading problem in MCC, firstly, the local and offloading costs for methods will be modeled based on before mentioned assumptions and models. The local time execution, $T_{i,local}$ for a service i can be determined by using the number of instructions I and mobile execution speed $Speed_{MN}$ as follows:

$$T_{i,local} = \frac{I}{Speed_{MN}} \tag{1}$$

The local energy consumption, $E_{i,local}$ for a service i can be determined by using the number of instructions I and mobile execution speed $Speed_{MN}$.

$$E_{i,local} = P_{i,local} * T_{i,local} \tag{2}$$

Where $P_{i,local}$ is the power for a local execution per second.
Here, x_i is introduced for a method i, which indicates whether the method i is executed locally or remotely(x_i =1 if a method i runs remotely and 0 if it runs local).By using x_i, the data sizes which will be sent, $D_{i,s}$, and recieved, $D_{i,r}$, are defined as follows.

$$D_{i,s} = send_i * x_i \tag{3}$$

$$D_{i,r} = rec_i * x_i \tag{4}$$

The time cost for offloading a method i to remote cloud, $T_{i,offload}$, can be expressed as the sum of taking time during waiting w period for getting results from the cloud and transferring time(including sending and receiving)as follows.

$$T_{i,offload} = (\frac{I}{Speed_{cloud}} + \frac{D_{i,s}}{B_{i,s}} + \frac{D_{i,r}}{B_{i,r}}) \tag{5}$$

where $Speed_{cloud}$ is the remote execution speed(cloud speed). $B_{i,s}$ and $B_{i,r}$ are the bandwidth for sending and receiving, respectively. The energy cost for offloading a method i to a remote cloud, $E_{i,offload}$, can be expressed as the sum of energy consumption during waiting period for getting results from a cloud $E_{i,idle}$, and transferring (including sending $E_{i,s}$ and receiving $E_{i,r}$) as follows.

$$E_{i,offload} = E_{i,s} + E_{i,idle} + E_{i,r} \tag{6}$$

where $E_{i,idle}$ can be expressed as multiplying of the idle time of the mobile device for waiting to get a result from the cloud, $t_{i,idle}$ and the power idle consumption per second $P_{i,idle}$.

$$E_{i,idle} = P_{i,idle} * t_{i,idle} \tag{7}$$

The energy consumption for sending data, $E_{i,s}$ can be expressed as multiplying of the time for sending data from mobile to cloud, $t_{i,s}$ and the power consumption for sending data from mobile to cloud per second, $P_{i,s}$.

$$E_{i,s} = P_{i,s} * t_{i,s} \tag{8}$$

The energy consumption for receiving data from cloud, $E_{i,r}$ can be expressed as multiplying of the time for receiving data from cloud $t_{i,r}$ and the power consumption for receiving data from cloud per second $P_{i,r}$.

$$E_{i,r} = P_{i,r} * t_{i,r} \tag{9}$$

The idle time of a mobile device during waiting period for getting a result from a cloud server can be treated as the execution time of a remote cloud, so $E_{i,offload}$ can be written as follows.:

$$E_{i,offload} = \frac{P_{i,idle} * I}{Speed} + \frac{P_{i,s} * D_i, s}{Bi, s} + \frac{P_{i,r} * D_i, r}{Bi, r} \tag{10}$$

By using Eq. 1 and Eq. 5, the total cost of execution time for n methods is

$$C_{time} = \sum_{i=1}^{n} (T_{i,local} * (1 - x_i) + T_{i,offload} * x_i) \tag{11}$$

Also, by using Eq. 2 and Eq. 10, the total cost of energy consumption for n methods is

$$C_{energy} = \sum_{i=1}^{n} (E_{i,local} * (1 - x_i) + E_{i,offload} * x_i) \tag{12}$$

Note that, the memory cost on the mobile device for n methods that run local can be calculated as follows.

$$C_{memory} = \sum_{i=1}^{n} mem_i * (1 - x_i) \tag{13}$$

In addition, the data transfer cost for a remote execution of n methods, that includes the transfer cost of its related methods which are not at the same execution location. If the output of one method is an input of another is determined by the following equation

$$C_{tranfer} = \sum_{i=1}^{n} cod_i * x_i + \sum_{i=1}^{n} \sum_{j=1}^{k} tr_i * (x_i XOR x_j) \tag{14}$$

where k is the number of related methods. By using Equations 11,12,13, and 14, the total overall cost for n methods can be written as :

$$C_{total} = C_{tranfer} * W_{tr} + C_{memory} * W_{mem} + C_{energy} * W_{energy} + C_{time} * W_{time} \quad (15)$$

where $W_{tr}, W_{energy}, W_{time}, W_{mem}$ are the weights of transferring, energy, time, and memory costs, respectively. These weights represent the importance of these costs in the offloading process such that the sum of these weights must equal 1 as follows.

$$W_{tr} + W_{mem} + W_{energy} + W_{time} = 1 \quad (16)$$

Here, the solution $x_1, x_2 ... x_n$ represents the required offloading partitioning of the application.
The objective goal of offloading problem in MCC is minimizing the overall cost C_{total} as much as possible such that takes into account the resources constraints of a mobile device. So, the objective function of this problem can be written as follows.

$$\min_{x \in 0,1} C_{total} \quad (17)$$

Such that

$$\sum_{i=1}^{n} mem_i * (1 - x_i) \leq avail_{memory} \quad (18)$$

$$C_{energy} \leq avail_{energy} \quad (19)$$

where constraint 18 means that the memory cost of a resident method can not be more than available memory on the mobile device. constraint 19 means that the energy cost of can not be more than available energy of the mobile device.

4. PROPOSED ALGORITHM

4.1 Basic Idea

In this section, to solve the offloading problem which was described and formulated in the previous section, the proposed offloading scheme will be introduced. The proposed scheme is called Application Method-Based Efficient Offloading Scheme (AMBEO).The basic idea of AMBEO is based on five issues: (1) dividing each mobile application into three layers: presentation layer, logical layer and data access layer, (2) considering each service or process in each layer as a set of methods, (3) the methods of presentation layer resides on the mobile device, (4) the methods of data layer is fully deployed on the cloud to minimize the data access, and (5) the methods of logic layer is distributed between a cloud server and a mobile device by using formulated cost model which was described in Section 3. Based on these five issues , the architecture of AMBEO is shown in Fig. 1.

4.2 Proposed Algorithm

AMBEO consists of two phases based on its basic ideas. The first phase is called *Profile phase* which determines the current value of a mobile, cloud server, each method in the logical layer, network conditions as bandwidth. While the second phase is called *Decision phase* which determines which method will be run on a mobile device or on a cloud based on the information that are calculated by using the profile phase, and the local and offloading costs as described in section 3. In the rest of this section, the detailed description of these two phases will be introduced.

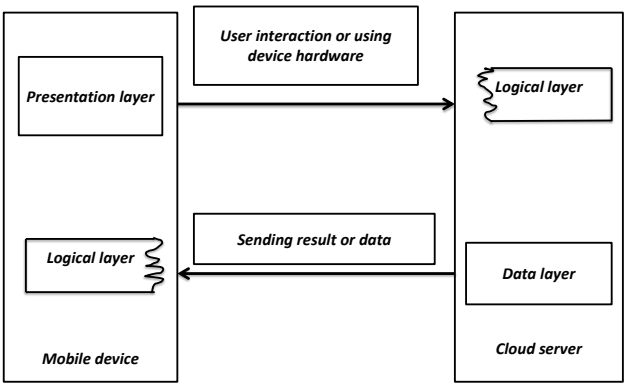

Fig. 1: Architecture of AMBEO

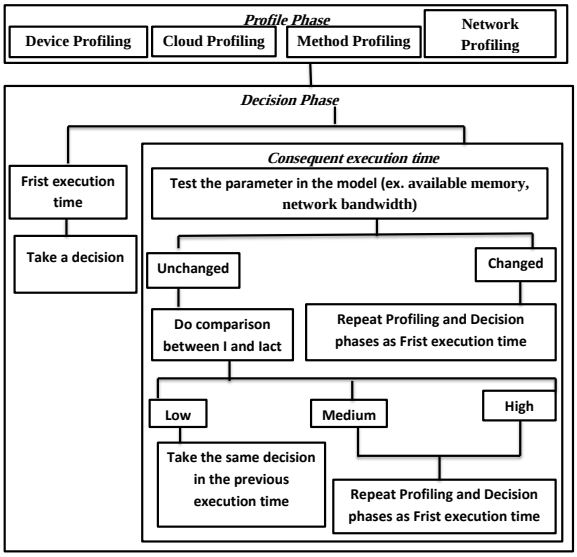

Fig. 2: AMBEO phases

4.2.1 Profile Phase. at runtime, before each method is invoked, AMBEO determines whether the method invocation should run locally or remotely. So, AMBEO measures the characteristics of a mobile device and a cloud server at the initialization time and it continuously monitors the network characteristics because these can often change and a stale measurement may force algorithm to make the wrong decision on whether a method should be offloaded or not. Therefore, the profile phase contains four profiling components: *Mobile device profiling, a cloud server profiling, a method profiling* and *a network profiling*. These four profiling components are determined and monitored as follows.

* **Device Profiling:** In this profiling, AMBEO determines an energy consumption of a mobile device by using a measurement application model for the energy consumption on a mobile device (e.g. Android phone) on the fly [11]. There is an online power estimation system that has been implemented for Android platform smartphones. PowerTutor provides accurate, real time power consumption estimates for power-intensive hardware components including CPU and LCD display as well as GPS, Wi-Fi, audio, and cellular interfaces[11, 12]. Also,Little Eye[13]

software used to measure the values of a processor speed, the available memory of a mobile, battery consumption, amount of data transfer and memory used.

* **Cloud Profiling:** In this profiling, AMBEO determines the value of a processor speed of a cloud server. AMBEO assumes that this value can be determined by using some means from the cloud.
* **Method Profiling:** In this profiling, AMBEO determines which the method belongs to a presentation layer, a logical layer or a data layer by using meta-data that stored in a manifest file and Java reflection such as OSGi[14] which has been traditionally used to decompose and loosely couple Java applications into software modules. Then, for each method that belongs to the logical layer it determines the characteristics of the methods such as code size, memory used and amount of data transferred by running a method on the device by using Little Eye Eyee[13] software.
* **Network Profiling:** In this profiling, AMBEO monitors the network and gathers all information about the network as Internet connection availability of a mobile device and the current bandwidth of the network to show its quality. This information can be obtained by using one of existing android APIs[15].

4.2.2 Decision Phase. In this phase, the offloading decision of a method depends on four factors:1) characteristics of mobile device 2) characteristics of the cloud 3) characteristics of methods such as input, output and code size 4) characteristics of network as network bandwidth.
For each method, there are two execution cases: *first execution time* case which means that this method will be run for the first time for a current running application and *consequent execution time* case which means that this method will be run for second time or more. These two cases are described as follows.

* **First execution time :** If a method will be run for the first time, AMBEO decides the following issues: (a) The methods of presentation layer will be run on a mobile device because there is a need for a user interaction or an access to a certain devices hardware. (b) The methods of data layer are fully deployed on the cloud if a remote server is available to minimize the data access. If a disconnect occurs, AMBEO resumes and runs the method on the mobile device, in this case, the application's energy consumption only incurs a small penalty cost due to offloading the method to the cloud. (c) The methods of logical layer will be run as follows:
(1) If a disconnect occurs, AMBEO runs the method on the mobile device. In this case, the application's energy consumption only incurs a small penalty cost due to offloading to the cloud. (2) If a cloud server is available and a memory cost of a method is larger than the available memory, the method will be run on the cloud. (3) If a cloud server is available and a memory cost of method is less than available memory, AMBEO executes the following steps: (i) calculate the time local and the energy consumption local costs by using Eqs. 1 and 2. Also, calculates the memory local cost for this method. (ii) calculates the time offloading and the energy consumption offloading costs by using Eqs. 5 and 10. Also, calculates data delay transfer cost which is represented by the sum of code size and receiving and sending data costs. (iii) if the total local cost is larger than the total offloading cost, the method will be run on the cloud otherwise the method will be run on the mobile device. (iv) saves the offloading decision (i.e., value of x_i) and the actual execution time of method, T_{act}. This actual execution time can be used to determine the actual number of instruction, I_{act}, of this method with respect to the number of instruction of a method which is defined in Eq. 1 or Eq. 5 according to x_i see Algorithm 1.
* **Consequent execution time:** This time means that a method will be run for second or more times. In this case, there are two cases of the model parameters as available memory or network bandwidth. (1) Changed case: this means that the values of these parameter are changed from their earlier values in the previous run. In this case, AMBEO repeats the two phases, Profiling and Decision phases as the first execution time.

Algorithm 1 AMBEO algorithm

input $S_{mobile}, S_{cloud}, P_s, P_{idel}, P_{local}, P_r, avail_{memory}, B, D_s, D_r$

for i=1 **to** i=n **do**
  if $mem_i > avail_{memory}$ **then**
    $x_i = 1$
  else
    $I \leftarrow$ getcodsizeofmethode(i)
    $T_{ilocal} \leftarrow \frac{I}{S_{mobile}}$
    $E_{ilocal} \leftarrow \frac{P_{local*I}}{S_{mobile}}$
    $T_{ioffload} \leftarrow (\frac{I}{S_{cloud}} + \frac{D_s}{B_s} + \frac{D_r}{B_r})$
    $E_{ioffload} \leftarrow \frac{P_{idle*I}}{S_{cloud}} + \frac{P_s*D_s}{B_s} + \frac{P_r*D_r}{B_r}$
    $C_{local} \leftarrow W_{time} * T_{local} + E_{local} * W_{energy} + mem_i * W_{mem}$
    $tr_i \leftarrow D_s + D_r$
    $C_{transfer} \leftarrow cod_i + tr_i$
    $C_{offload} \leftarrow W_{time} * T_{offload} + E_{offload} * W_{energy} + C_{transfer} * W_{tr}$
    if $C_{offload} < C_{local}$ **then**
      $x_i = 1$
    else
      $x_i = 0$
    end if
  end if
  if $x_i = 0$ **then**
    $I_{act} = \frac{I*T_{act}}{T_{local}}$
  else
    $I_{act} = \frac{I*T_{act}}{T_{offload}}$
  end if
end for

(2) Unchanged case: this means that the values of these parameters are not changed from their earlier values in the previous run. In this case, AMBEO compares the number of instructions of a method, I (which determine in method proling step) and the actual number of instructions, I_{act} to determine the changing degree of these parameters. If the result is in the period [0.0,0.2], the change is called Low and if the result is in the period [0.21,0.6] the change is called Medium. Otherwise the change is called High. In case of Low change, AMBEO takes

the same decision in the previous execution time. while, in case of Medium or High AMBEO repeats the two phases of AMBEO as the first execution time as shown in Fig. 2.
Algorithm 1 shows the steps of AMBEO scheme. In Steps 2 and 3, AMBEO compares the needed memory cost and the available memory for n methods in the logical layer. In Steps 5 to 13, AMBEO calculates the cost for running a method on the mobile device and the cost for running a method on the cloud server if the memory cost is more than the available memory. In Steps 14 to 19, AMBEO compares the cost for running a method on the mobile device and the cost for running a method on the cloud server and determines the value of x_i which indicates the offloading of a method to the cloud or not. In Steps 20 to 25, AMBEO saves the information for future decisions.

5. SIMULATION RESULTS AND QUALITATIVE EVALUATIONS

In this section, the simulation results will be presented to show the offloading performance of running a method by using AMBEO comparing to performance of running it on a mobile locally. Also, the qualitative comparison between AMBEO and other offloading schemes will be presented in the end of this section.

5.1 Simulation Outline

Firstly, AMBEO is implemented by using c++ programing language. Due to the difference between the methods of mobile cloud application and network bandwidth, different scenarios are generated for method input, method code size, method output and network bandwidth (four scenarios). The simulation parameters as mobile processor speed, cloud processor speed, consumed power by mobile in ideal case, and consumed power by mobile for sending and receiving data are shown in Table 1.

Table 1. : Simulation Parameters.

Parameter	Value
Mobile processor speed	0.6 GHz
Cloud processor speed	2.8 GHz
Mobile available ram	256 MB
Consumed power by mobile in ideal case	0.89 J
Consumed power by mobile for sending data	1.6 J
Consumed power by mobile for receiving data	1.6 J

5.2 Simulation Results and Analysis

* ***Scenario A:*** In this scenario, AMBEO uses the values for method input, method code size, method output and network bandwidth as shown in Table 2 to compare the cost of a running method locally and the cost of running method by using AMBEO. Fig -3 shows the cost of running method on local device and by using AMBEO against the network bandwidth. As shown in Fig. 3, the cost of AMBEO and locally running are equal when the bandwidth is less than or equal to 700 Kbps. While the cost of AMBEO decreases as bandwidth increases and is much lower than the cost of running local when the bandwidth is larger than 700 Kbps. This is because, AMBEO considers the cost transfer which decreases as the bandwidth increases. as a result, the total cost of running method remotely decreases.

Table 2. : Scenario A: cost parameters Vs. network bandwidth.

Parameter	Value
Method input	20 Kb
Method output	100 Kb
Code size of method	2000 Kb
Bandwidth	10-1500 Kbps

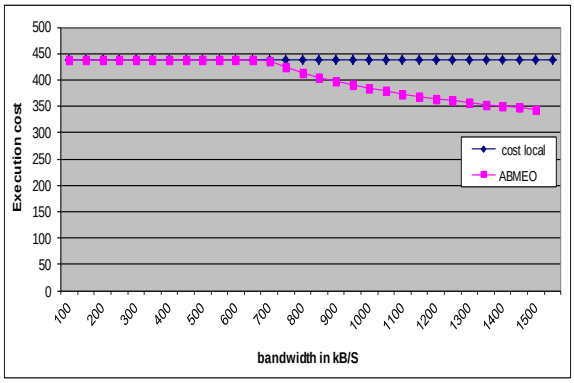

Fig. 3: cost vs network bandwidth

* ***Scenario B:*** In this scenario, AMBEO uses the values for method input, method code size, method output and network bandwidth as shown in Table 3 to compare the cost of a running method locally and the cost of running method by using AMBEO. Fig. 4. shows the cost of running method on local device and by using AMBEO against the code size. As shown in Fig. 4, the cost of AMBEO and locally running increases as codes size increases. Also, the two costs are equal when the bandwidth is less than or equal to 600 Kbps. While the cost of AMBEO is lower than the cost of running local when the code size is larger than 700 Kbps. This is because, when the code size of method is high (i.e., number of instructions of method is high) and the processor speed of cloud is higher than the processor speed of mobile device, the cost of running on the cloud is less than the cost of local running.

Table 3. : Scenario B: cost parameters Vs. Method cod size.

Parameter	Value
Method input	20 Kb
Method output	100 Kb
Code size of method	100-1250 Kb
Bandwidth	3 Mbps

Table 4. : Scenario C: cost parameters Vs. Method output.

Parameter	Value
Method input	20 KB
Method output	0-390 KB
Code size of method	1000 KB
Bandwidth	3 Mbps

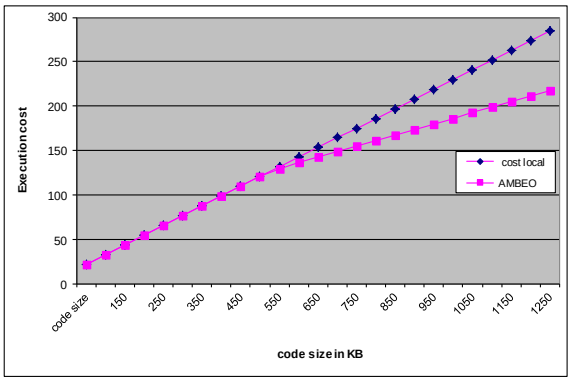

Fig. 4: cost vs code size of method

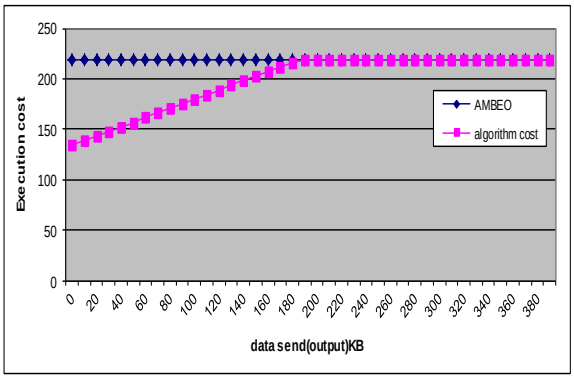

Fig. 5: cost vs data send(method output)

* ***Scenario C:*** In this scenario, AMBEO uses the values for method input, method code size, method output and network bandwidth as shown in Table 4 to compare the cost of running method locally and the cost of running method by using AMBEO. Fig. 5. shows the cost of running method on local device and by using AMBEO against the data send size. As shown in Fig. 5, the cost of AMBEO increases as data send size increases. While the cost of local running is fixed. Also, the cost of AMBEO is much less than the cost of local running when the data send size is less than or equal to 190 kb. While the two costs are equal when the data send size is larger than 190 kb. This is because, AMBEO decides that running method must run on the local device when the size of data send is high then the cost of time offloading, energy offloading, and transfer will be increase.

Table 5. : Scenario D: cost parameters Vs. Method input.

Parameter	Value
Method input	0-190 KB
Method output	100 KB
Code size of method	1000 KB
Bandwidth	3 Mbps

* ***Scenario D:*** In this scenario, AMBEO uses the values for method input, method code size, method output and network bandwidth as shown in Table 5 to compare the cost of running method locally and the cost of running method by using AMBEO. Fig. 6.

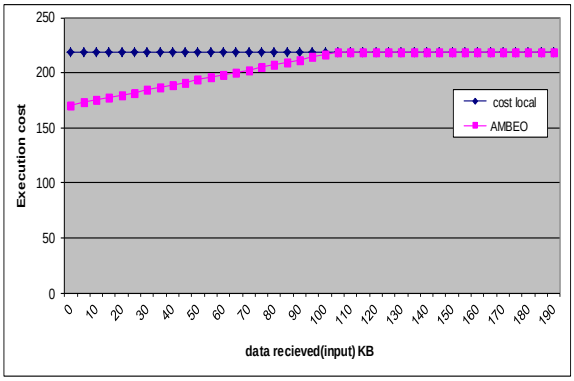

Fig. 6: cost vs data received(method input)

shows the cost of running method on local device and by using AMBEO against the data received size. As shown in Fig. 6, the cost of AMBEO increases as data received size increases. While the cost of local running is fixed. Also, the cost of AMBEO is much less than the cost of local running when the data received size is less than or equal to 110 kb. While the two costs are equal when the data received size is larger than 110 kb. This is because, AMBEO decides that running method must run on the local device when the size of data received is high then the cost of time offloading, energy offloading, and transfer will be increase..

5.3 Qualitative Comparison

In this section, the qualitative between AMBEO and some of existing approaches according to the following criteria:

Adaptive with network change (ANC): This means that the approaches continuously monitors the network and is adaptive with network disconnection.

Saving energy (SE): This means that a cost model (optimization problem) includes the parameter of energy consumption cost .

Memory cost (MC): This means that a cost model (optimization problem) includes the parameter of memory cost and takes into account the available memory of the mobile device.

Offloading level (OffL): Usually, from the view of developer or programmer, the tasks of an application or a service can be viewed as a set of classes objects, threads, software modules , or methods. So, the offloading level means that the offloaded entities which is needed to moved into the cloud are class objects, threads, software modules, or methods.

Prediction of second execution (PSE): This means that the algorithm can predict the second execution or not.

Minimize data transfer (MDT): this means that a cost model (optimization problem) includes the amount of data transfer to avoid data traffic.

According to the qualitative parameter, the best criteria is as follows: ANC is *Yes*, SE is *Yes*, MC is *Yes*, OffL is *methods*, PSE is *Yes*, and MDT is *Yes*. The qualitative evaluation is shown in Table 6. As shown in Table 6, AMBEO satisfies all requirements of the best criteria among existing approaches.

Table 6. : Qualitative Comparison

	ANC	SE	MC	OffL	PSE	MDT
[4]	No	No	No	threads	No	NO
[6]	Yes	No	Yes	software modules (bundles)	No	Yes
[5]	Yes	Yes	No	methods	Yes	No
AMBEO	Yes	Yes	Yes	methods	Yes	Yes

6. CONCLUSION

In this paper, the offloading problem in mobile cloud computing and a lot of studies have been appeared to support remote execution for mobile applications on the cloud are introduced. In addition, a new offloading algorithm called AMBEO is proposed. AMBEO provides method level code offloading which improves the performance and save energy of the mobile device. AMBEO can decide which method will run in local device or must be offloaded into the cloud based on the cost of its running. The performance of AMBEO algorithm has been evaluated through extensive simulation with different values of network bandwidth, method input, method cod size, and method output. The simulation results demonstrated that AMBEO is better than existing approaches in reducing the total cost of running an application or a service. The future work will focus on trying to propose new application model for making the offloading decision that considers the enabling parallelization on the cloud.

7. REFERENCES

[1] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[2] Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokhar, and Rajkumar Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(3):1294–1313, 2013.

[3] A Khan, Mazliza Othman, S Madani, and S Khan. A survey of mobile cloud computing application models. 2013.

[4] 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*, pages 301–314. ACM, 2011.

[5] 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*, pages 49–62. ACM, 2010.

[6] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Middleware 2009*, pages 83–102. Springer, 2009.

[7] Dejan Kovachev, Tian Yu, and Ralf Klamma. Adaptive computation offloading from mobile devices into the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 784–791. IEEE, 2012.

[8] Jan S Rellermeyer, Gustavo Alonso, and Timothy Roscoe. R-osgi: distributed applications through software modularization. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, pages 1–20. Springer-Verlag New York, Inc., 2007.

[9] Jan S Rellermeyer, Oriana Riva, and Gustavo Alonso. Alfredo: an architecture for flexible interaction with electronic devices. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 22–41. Springer-Verlag New York, Inc., 2008.

[10] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[11] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, 2010.

[12] http://powertutor.org/.

[13] http://www.littleeye.co/.

[14] OSGi Alliance. Osgitm service platform, core specification, release 4, version 4.1, 2007.

[15] https://developer.android.com/reference/android/net/networkcapabilities

[16] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[17] Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *Communications Surveys & Tutorials, IEEE*, 14(4):1232–1243, 2012.

[18] Muhammad Shiraz, Md Whaiduzzaman, and Abdullah Gani. A study on anatomy of smartphone. *Computer Communication & Collaboration*, 1:24–31, 2013.

[19] Xinwen Zhang, Sangoh Jeong, Anugeetha Kunjithapatham, and Simon Gibbs. Towards an elastic application model for augmenting computing capabilities of mobile platforms. In *Mobile wireless middleware, operating systems, and applications*, pages 161–174. Springer, 2010.