Task Offloading with Execution Cost Minimization in Heterogeneous Mobile Cloud Computing

Xing Liu[1], Songtao Guo[1(✉)], and Yuanyuan Yang[2]

[1] College of Electronic and Information Engineering, Southwest University, Chongqing 400715, China
songtao_guo@163.com

[2] Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA

Abstract. Mobile cloud computing (MCC) can significantly enhance computation capability and save energy of smart mobile devices (SMDs) by offloading remoteable tasks from resources-constrained SMDs onto the resource-rich cloud. However, it remains a challenge issue how to appropriately partition applications and select the suitable cloud to offload the task under the constraints of execution cost including completion time of the application and energy consumption of SMDs. To address such a challenge, in this paper, we first formulate the partitioning and cloud selection problem into execution cost minimization problem. To solve the optimization problem, we then propose a system framework for adaptive partitioning and dynamic selective offloading. Based on the framework, we design an optimal cloud selection algorithm with execution cost minimization which consists of offloading judgement and cloud selection. Finally, our experimental results in a real testbed demonstrate that our framework can effectively reduce the execution cost compared with other frameworks.

Keywords: Mobile cloud computing · Application partition
Task offloading · Cloud selection · Execution cost minimization

1 Introduction

In recent years, smart mobile devices (SMDs) such as smartphones have become an indispensable part of modern life. The SMDs have been the preferred computing device to accommodate most up-to-date mobile applications, like interactive games, image/video applications and etc [16]. However, due to the physical size constraint, SMDs are in general resource-constrained [7], with limited energy supply and computation capacity. In particular, it is still a challenge how to run computing-intensive applications on resource constrained SMDs.

With the development of communication technology, cloud computing applied in the mobile industry has formed an emerging and promising method to solve this challenge [7], which is mobile cloud computing (MCC). MCC allows mobile devices to take advantage of rich resources provided by the clouds. Thus, MCC not only extends battery lifetime but also utilizes the computation resource of cloud system. In recent years, a *computation offloading* [7,14] which migrates resource-intensive computations from SMDs to the cloud via wireless access, has been proposed as a way of implementing mobile cloud computing. Moreover, the suitable *computation partition* is precondition of computation offloading, which also is the hot research topic of MCC. The objective of computation partition and offloading is solving the problem which minimizes the execution cost of applications including completion time and energy consumption of SMDs.

Previous research works have proposed solutions to address the problem [5,6,8,12,14–16]. Chun et al. in [14] proposed a CloneCloud that automatically offloads an application from the mobile device to the smartphone clone in the cloud at a fine-granularity level while optimizing the execution time for a target computation. Based on CloneCloud, Yang et al. in [16] optimized the overall execution time by dynamically offloading a part of Android codes running on smart mobile device to the cloud. In practice, according to computing resources, the clouds can be divided into many categories. For different categories of clouds, the offloading cost of the same application is different. However, how to select an appropriate category of clouds to minimize the execution cost is not considered in the previous works.

This paper mainly focuses on how to appropriately partition application and dynamically select the best cloud to offload. First, we formulate the cloud selection problem into an optimization problem of minimizing execution cost, which includes the completion time of application and energy consumption of SMDs. In order to solve this problem, we then design a novel system framework which performs the method-level offloading with least transfer package size. This framework provides runtime support for the application partitioning and offloading, and consists of profiler, solver and communication module. According to the amount of local computation resource, the SMDs divide each thread of the application into some small tasks, named offloading tasks, and migrates the tasks to the best cloud to execute so as to achieve the minimum execution cost. Based on the framework, we propose a best cloud selection algorithm assigned into the solver of framework.

Compared with previous works, the contributions of this paper can be summarized as follows:

– We present an integrated and novel framework of code partitioning and offloading. The comments of our framework are highly modularized and easily extended.
– We propose an optimization model for the local execution time and energy consumption of SMDs by taking into account the execution time of the cloud.
– Based on the proposed optimization model, we provide a cloud selection algorithm to achieve the minimum execution cost.

The rest of this paper is organized as follows. In Sect. 2, we introduce the related work. In Sect. 3, we present the MCC system and optimization model.

Section 4 outlines the proposed framework and Sect. 5 presents the algorithms for optimal cloud selection. In Sect. 6, we evaluate the performance of the proposed algorithm. Section 7 concludes the paper.

2 Related Work

MCC focuses on solving the problems of *what* to offload and *how* to offload [10,11]. The primary objective of offloading and partitioning policies is to enhance the performance of mobile device in terms of execution/completion time and throughput by utilizing cloud resource [4,7,13–16]. Guo et al. in [7] proposed an energy-efficient dynamic offloading strategy that optimizes the performance of mobile devices through the dynamic voltage and frequency scaling (DVFS) in local computing. Yang et al. in [15] studied the computation partitioning in order to optimize the partition between the mobile devices and cloud such that the application has maximum throughput.

In addition, in [14], the Multi-User Computation Partitioning Problem (MCPP) was designed to achieve minimum average completion time for all the users. Compared with these works, we not only consider the cost of thread offloading, but also take into account the price of thread partition. On the basis of the partition technique in [16], we study how to judge whether an application thread is necessary to offloading. After that, we develop heterogeneous selection scheme for diverse remote cloud resources with the optimal size of the transmission packet.

There are a few works on the design of application frameworks in cloud computing [5,6,8,12,13,15]. The most popular one is MAUI [6], which describes a system which offloads fine-grained code to the cloud, while maximizing the potential of energy saving. However, it can't guarantee to satisfy the requirement of completion time. Actually, either completion time or energy consumption was only considered in previous works. Our work aims to develop systematic method to improve the time and energy efficiency of task offloading. Thus we utilize an online profiler to monitor the completion time of the application, and dynamically decide whether and where to offload tasks based on user requirements.

3 Network Architecture and Problem Formulation

3.1 Network Architecture

Our cloud system consists of a specific SMD and multiple types of mobile cloud systems that can provide different services, as shown in Fig. 1. The SMD accesses the Internet via base station or wireless access point, and then visits the cloud resources over the network.

In this paper, two kinds of cloud system would be considered, i.e., central cloud c_1 and cloudlets c_2. Furthermore, $c_k = \{c_k^1, c_k^2, \cdots, c_k^m\}$, $k = 1, 2$ denotes the cloud c_k has m cloud severs. However, the CPU frequencies of different servers in a cloud system are different. Thus, the CPU frequency of each sever in cloud c_k is $f_{c_k}^j$,

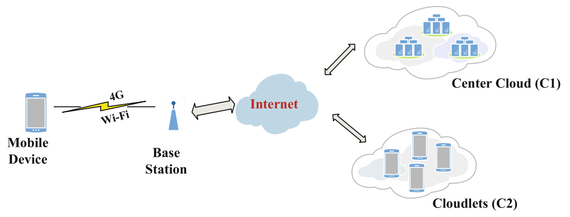


Fig. 1. Overview of mobile cloud computing system

where $j \in \{1, 2, \cdots, m\}$ represents the sever j of cloud c_k. Note that f_0 denotes the local CPU frequency of SMDs. In addition, we denote when the SMD accesses to cloud c_k via network access, the corresponding channel data transmission rate is $R_{c_k}, k = 1, 2$.

3.2 Problem Formulation

MCC aims to solve the problem of energy consumption and execution time of SMDs, which is the main research content of this paper.

In our system, a process is an Android application running on the Dalvik virtual machine (VM). An application process may comprise multiple threads, part of which are called as *remoteable threads*, which may contain multiple remotely executable methods (REMs) while others will be called as *un-remoteable threads*, which do not have REMs. MCC mainly focuses on the remoteable threads. Thus unless otherwise specified, the thread in this paper is considered as the migratable thread.

Next, we consider the execution cost of an application under the diversity condition. When the SMD runs an application, the solver in our framework will partition the application as primary heap objects (PHOs), which will be described in Sect. 4.2.1 in detail. We define the set of PHOs as $N = \{i|i = 1, 2, ..., n\}$. Moreover we leverage an indicator $x_{i,c_k}, k = 1, 2, \forall i \in N$, to represent the task allocation, i.e.,

$$x_{i,c_k} = \begin{cases} 1, & \text{if task}\, i\, \text{is assigned to cloud}\, c_k, \\ 0, & \text{otherwise.} \end{cases}$$

where x_{i,c_k} is either 0 or 1, thus we can take $X = \{(x_{i,c_k})|i \in N, k = 1, 2\}$ as task allocation matrix.

We use a tuple $\{\alpha_i, \omega_i\}$ to denote task i, for $i \in N$, in which α_i is the input data size (in bits) from SMDs to cloud, and ω_i is the number of CPU cycles that is required by task processing, respectively. For a task i, we consider whether offloading it or not from the aspect of completion time and energy consumption. Similar to the existing work [4], we ignore the download time and downlink energy consumption of the task. Therefore, the completion time of task

T_i includes the computing time T_i^{comp} and transmission time T_i^{trans}, formulated as (1).

$$T_i = T_i^{comp} + T_i^{trans} \quad = \frac{\omega_i}{f_{c_k}^j} + \frac{\alpha_i}{R_{c_k}} \tag{1}$$

where $k = 1, 2$, and $T_i^{loc} = \frac{\omega_i}{f_0}$ denotes the local execution time of task i.

The energy consumption of task i, denoted as E_i, consists of two parts, which are the energy consumption of waiting for remote execution E_i^{wait}, and the energy consumption of transferring task to clouds E_i^{trans}, described as

$$E_i = E_i^{wait} + E_i^{trans} \quad = P_{idle} \times \frac{\omega_i}{f_{c_k}^j} + P_s \frac{\alpha_i}{R_{c_k}} \tag{2}$$

where $k = 1, 2$, and P_{idle} indicates the waiting power of SMDs when task i is migrated to clouds. P_s denotes transfer power of SMDs. We let $E_i^{loc} = P_c \times \frac{\omega_i}{f_0}$ denote the local computing energy consumption when task i is executed locally, where P_c represents computation power of SMDs.

Furthermore, the execution time $T(X)$ of an application can be expressed as Eq. (3).

$$\begin{aligned} T(X) &= \sum_{i \in N} x_{i,c_k} \times T_i \\ &= \sum_{i \in N} x_{i,c_k} \times \left(\frac{\omega_i}{f_{c_k}^j} + \frac{\alpha_i}{R_{c_k}} \right) \quad = \sum_{i \in N} x_{i,c_k} \varTheta_{i,c_k} \end{aligned} \tag{3}$$

where $k = 1, 2$. The energy consumption $(E(X))$ can be given by Eq. (4)

$$\begin{aligned} E(X) &= \sum_{i \in N} x_{i,c_k} \times E_i \quad = \sum_{i \in N} x_{i,c_k} \times \left(P_{idle} \frac{\omega_i}{f_{c_k}^j} + P_s \frac{\alpha_i}{R_{c_k}} \right) \\ &= \sum_{i \in N} x_{i,c_k} \varPhi_{i,c_k} \end{aligned} \tag{4}$$

In particular, the local execution time and energy consumption of application are given by respectively

$$T^{loc}(X, f_0) = \sum_{i \in N} x_{i,0} \frac{\omega_i}{f_0} \tag{5}$$

$$E^{loc}(X, f_0) = \sum_{i \in N} x_{i,0} P_c \times \frac{\omega_i}{f_0} \tag{6}$$

We use the execution cost as the metric to measure whether to migrate to the cloud. Hence, execution cost $Cost(X)$ can be defined by the summation of makespan $T(X)$ and the energy consumption $E(X)$ of the application, for $k = 1, 2$, i.e.,

$$\begin{aligned} Cost(X) &= \lambda_t T(X) + \lambda_e E(X) \\ &= \lambda_t \sum_{i \in N} x_{i,c_k} \varTheta_{i,c_k} + \lambda_e \sum_{i \in N} x_{i,c_k} \varPhi_{i,c_k} \end{aligned} \tag{7}$$

where $\lambda_t, \lambda_e \in [0,1]$ are scalar weights, and $\lambda_t + \lambda_e = 1$. These weights can be adjusted by the preference related with energy and delay deadline of users.

Overall, the optimization problem formulated is how to select the cloud to offload for minimizing the execution cost of the task. The optimization framework can be formulated as follows:

$$\min_{X,C,S} Cost(X) \tag{8}$$

$$s.t \begin{cases} \sum_{i \in N} T(X) \le t_{delay} & \text{(9a)} \\ \sum_{i \in N} E(X) \le e_{threshold} & \text{(9b)} \\ \sum_{k=1,2} x_{i,c_k} = 1, \quad \forall i \in N & \text{(9c)} \\ X = \{(x_{i,c_k}) | i \in N, k = 1, 2\} & \text{(9d)} \end{cases}$$

The constraint (9a) denotes the completion time constraint which ensures that the total completion time of all the tasks in an application executed on SMDs is bounded by the required maximum finish time (i.e., delay deadline), t_{delay}. Similarly, (9b) specifies that the total energy consumption is less than or equal to the maximum energy consumption $e_{threshold}$. Constraint (9c) demonstrates that a task can only be assigned to one device. (9d) represents the task allocation.

4 System Framework Design

To address the optimization problem (8), a system framework is proposed, which consists of three components: Profiler, Solver and Communication Module, shown in Fig. 2. First, the profiler is mainly responsible for analyzing and

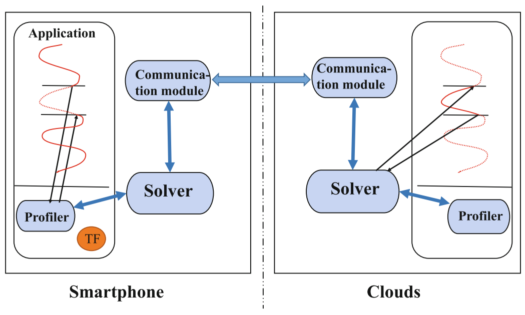


Fig. 2. The overview of system framework

detecting the network conditions, cloud conditions and mobile device performance, as well as transmitting correlative data to the solver. Then, based on the data provided by the profiler, the solver makes decision on partitioning and offloading. Finally, the communication module sends the task packages to the cloud, and receives the results returned from the cloud. In the following, we introduce the three components in details.

4.1 Profiler

The profiler is mainly deployed on the mobile device to detect and collect the parameters and configure resources of SMDs and clouds.

When SMD starts to execute an application, the system would create a temporary buffer (TF) in SMD's memory to store the information consisting of the device performance, such as P_{idle}, P_s, and P_c of SMDs, the network access, i.e., the channel data transmission rate R_{c_k}, and the cloud resource, that is, the types of clouds c_k, as well as the CPU frequency $f_{c_k}^j$. Moreover, this temporary buffer will be immediately released once the application is completed.

The solver can directly get data from cache *TF*, when it makes decision. What's more, the profiler is detecting the information in real-time, updating the data of *TF* at any time to ensure that the values obtained by solver are not expired.

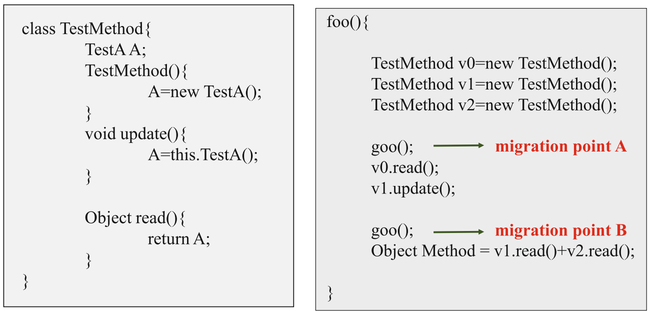
Furthermore, the profiler also measures and analyzes whether the thread of the mobile application is remoteable. Here, we define three types of unremoteable codes: (1) the codes that implement the application's user interface; (2) the codes that interact with I/O devices where such interaction only makes sense on the mobile device; (3) the codes that interact with any external component that would be affected by re-execution [6]. If the thread is remoteable, which indicates that the thread doesn't include above three types of codes, the profiler will mark this thread as attribute [Remoteable], and transfer it to the solver.

4.2 Solver

The solver is mainly used to solve the problem of minimizing the execution cost $Cost(X)$ of an application, which consists of two modules in our system: the partition module and the migration module.

4.2.1 Partition Module

Partition module is used for code partition of application. When the profiler detects a remoteable thread, the partition module catches this thread. The state being transferred of remoteable thread includes stack, register and reachable heap object. Therefore, by using the partitioning technique in [16], the partition module determines the accessible heap objects (AHOs) by recursively chasing the reference links. Then the partition module deletes the super classes to make AHOs become the primary heap objects (PHOs), which is fewer than AOHs.



```
class TestMethod{
        TestA A;
        TestMethod(){
                A=new TestA();
        }
        void update(){
                A=this.TestA();
        }

        Object read(){
                return A;
        }
}
```

(a) Class definition of TestMethod.

```
foo(){

    TestMethod v0=new TestMethod();
    TestMethod v1=new TestMethod();
    TestMethod v2=new TestMethod();

    goo();      ——→  migration point A
    v0.read();
    v1.update();

    goo();      ——→  migration point B
    Object Method = v1.read()+v2.read();

}
```

(b) Method declaration of foo.

Fig. 3. A example of Java code.

After that, we use the notion of dirty to further partition the PHOs. An example of code is shown in Fig. 3. Figure 3(a) defines the class $TestMethod$, and Fig. 3(b) declares the method $foo()$ and $goo()$ when these methods call the class $TestMethod$. The method $foo()$ calls method $goo()$ twice in Fig. 3(b), where we consider that two call points of the method $goo()$ are migration point A and migration point B, respectively. When there is the migration point A, class objects $V0, V1, V2$ are not invoked. While there is migration point B, class objects $V0, V1$ have been called, thus we label the objects, $V0, V1$, as dirty that have been invoked before call points. Other objects like $V2$ are un-dirty.

The dirty objects are identified by a famous compiler analysis technique, called *side-effect* [16]. For the un-dirty object, we do not migrate it rather than create a stub, and only migrate the stub. The stub consists of class name, object ID, and the address of an object which is necessary for the solver on the cloud to create new instance of un-dirty PHOs. After migrating stub to the cloud, we would use *on-the-fly* technique [3] to online instantiate un-dirty PHOs, named *on-cloud-copy*. Finally, we migrate the dirty objects and the stubs of un-dirty objects of PHOs, which is called offloading task.

4.2.2 Migration Module

The migration module mainly performs offloading decision for the task. In this module, we propose a best cloud selection algorithm (introducing in Sect. 5). According to transmission delay and energy saving required by the mobile user's preference for application execution, the module makes optimal offloading decision, i.e. selecting a best cloud to execute the migrated tasks, to minimize the cost $Cost(X)$.

We adopt t_{delay} to denote the tolerance of execution delay and $e_{threshold}$ to represent the tolerance of energy consumption. Different offloading strategies can be made by users based on their requirements for energy and delay as follows:

– When a mobile device is at low battery energy state, it can choose $\lambda_t < \lambda_e$, where $\lambda_t, \lambda_e \in [0, 1]$. Meanwhile $\sum_{i \in N} E(X)$ is bounded by $e_{threshold}$, i.e., $\sum_{i \in N} E(X) \leq e_{threshold}$.

- When a mobile device is running delay-sensitive applications (e.g., video streaming) that require to reduce as much as delay, it can set $\lambda_t > \lambda_e$, where $\lambda_t, \lambda_e \in [0,1]$. Simultaneously, $\sum_{i \in N} T(X) \leq t_{delay}$.
- When a mobile device has low battery energy and runs the delay-sensitive applications, it can set $\lambda_t = \lambda_e$, where $\lambda_t, \lambda_e \in [0,1]$, so as to jointly optimize the energy consumption of mobile devices and the application completion time. Similarly, $\sum_{i \in N} E(X) \leq e_{threshold}, \sum_{i \in N} T(X) \leq t_{delay}$.

4.3 Communication Module

Communication module is responsible for the communication between local mobile device and clouds. When the solver has partitioned the thread and made offloading decision, the communication module serializes and packages the offloaded states and sent to cloud. When the cloud finishes the execution of a task, the communication module at cloud serializes the execution result, and sends it to mobile device. The local communication module receives the results from the cloud, and then compares it with source codes. The results from the cloud are merged with the local source codes and the SMD run the merged program again.

In general, our system framework analyzes an application via profiler to determine whether each thread of the application can be offloaded. Furthermore, using the partition module of solver of the framework, these threads that need to be uploaded are partitioned as a smaller size of tasks, while the migration module calculates the execution time T_i, the energy consumption E_i, and the execution cost of each task $Cost_i$, according to the user's different preferences for time and energy consumption. We propose a best cloud selection algorithm on the migration module of solver based on the computation result of the partition module and obtain the cloud category with least cost, which will be given in Sect. 5. Finally, we offload the task to the selected cloud.

5 Best Cloud Selection Algorithm Design

In this section, we design a best cloud selection algorithm shown in Algorithm 1, which selects a cloud to transfer the task for achieving the minimum execution cost $Cost(X)$. The algorithm is applied to the migration module of our framework and consists of two parts: one is to decide whether the task can be offloaded, and the other is to migrate the remoteable task to which clouds.

In the following, we describe the first part of our algorithm. When the solver captures a task execution thread, we need to determine whether the thread needs to be uploaded to the cloud. Except for the previous judgement of profiler, we also need to decide whether the partitioned task is offloaded or not according to T_i and E_i. The judgement condition is given as Eq. (10).

$$\begin{cases} \text{task i is executed remotely,} & if\ \frac{T_i^{loc}}{T_i} > 1, and\ \frac{E_i^{loc}}{E_i} > 1 \\ \text{task i is executed locally,} & otherwise. \end{cases} \tag{10}$$

For a given task i, if its local computation time T_i^{loc} is greater than the remote execution time T_i, and the local energy consumption E_i^{loc} is also larger than the offloading energy consumption E_i, then the task will be offloaded to the cloud; otherwise, the task will be executed on local device. If task i is considered to be migrated, then the algorithm labels the attribute [*Remoteable*] to the task, as described in line 2–10 of Algorithm 1.

Besides, we will describe the second part of our algorithm. As described in line 14–15, the task i is determined to be uploaded to the cloud, and the cost $Cost_i$ of the task i for cloud c_1 and c_2 is calculated, as shown in line 14. Next, we compare all $Cost_i$ of task i by line 15, and obtain the minimum cost $Cost_i^{min}$. After that, we assign the cloud c_k with minimum cost $Cost_i$ of task i to the best cloud K. Finally, we select the cloud K as the best offloading cloud, and migrate the partitioned task to the cloud, as shown in line 16–18.

In addition, we analyze the time complexity of the algorithm. We consider n tasks and m clouds in the algorithm. For each task, the time complexity of calculating minimum cost of finding the best cloud among m clouds is $O(m)$, which is shown in Lines 12–20. Therefore, the time complexity of n tasks to calculate the minimum cost is denoted as $O(n \times m)$, from Lines 2 to 21.

Algorithm 1. Best Cloud Selection Algorithm.

```
Input: : $\lambda_t, \lambda_e$: user's preference;
    $i \in N$: execution tasks that had partitioned;
    $j = 1, 2, \cdots, m$: the number of severs of $c_k$;
Output: : best cloud selection $K$;
 1: set parameters: $f_{c_k}^j, R_{c_k}, P_{idle}, P_s, P_c$
    and $T_i, T_i^{loc}, E_i, E_i^{loc}, Cost_i, Cost_i^{min}$;
 2: for $i = 1$ to $n$ do
 3:     compute $T_i$, $T_i^{loc}$ and $E_i$, $E_i^{loc}$ by (1), (2), (5) and (6) respectively;
 4:     /* offloading judging */
 5:     if $\frac{T_i^{loc}}{T_i} > 1$ and $\frac{E_i^{loc}}{E_i} > 1$ then
 6:         task $i$ will be migrated to clouds;
 7:         label $i$ as [Remoteable];
 8:     else
 9:         execute task $i$ locally;
10:     end if
11:     /* cloud selection */;
12:     if task $i$ is remoteable then
13:         for $j = 1$ to $m$ do
14:             $Cost_i = \lambda_t T_i + \lambda_e E_i$; //$Cost_i$ is the execution cost of a task $i$ on cloud $c_k$
15:             $Cost_i^{min} = \min_{c_k \in C}\{Cost_i\}$
16:             if $K = c_k$ then
17:                 migrate task $i$ to cloud $c_k$;
18:             end if
19:         end for
20:     end if
21: end for
```

6 Performance Evaluation

We implement our system module on the Android 4.1.2. The smart mobile device is a SAMSUNG Galaxy Nexus with dual-core 1.2 Ghz CPU and 1 GB of RAM. As for the cloud sever, we consider two categories of clouds, i.e., the central cloud and cloudlets. The central cloud consists of 3 IBM X3850X6 severs, each of which has 4 quad-core 3.4 Ghz Xeon CPUs and 128 GB of RAM running Ubuntu 14.0. Then we use 30 Android 4.1.2 SAMSUNG Nexus S5 smartphones with quad-core 2.5 GHz CPU and 2 GB RAM, as cloudlets. The experimental parameters are listed in Table 1.

Table 1. Default parameter setup

Parameter	Value
ω_i	$330\alpha_i$
$f_{c_k}^j$	[10, 54.4] GHz
f_0	2.4 GHz
R_{s_v,c_k}^{UL}	[10, 25] Mbps
R_{s_v,c_k}^{DL}	R_{s_v,c_k}^{UL}
P_s	1.5 W
P_c	2.4 W
P_{idle}	50 mW

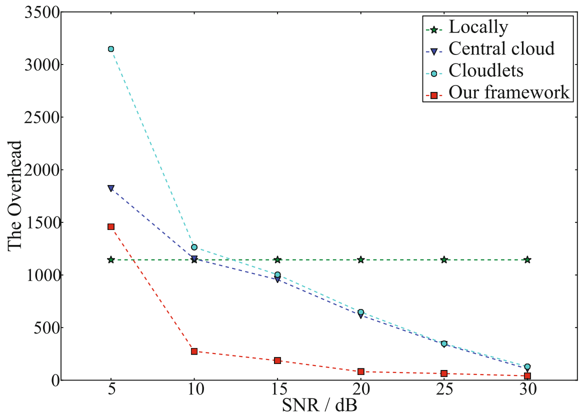
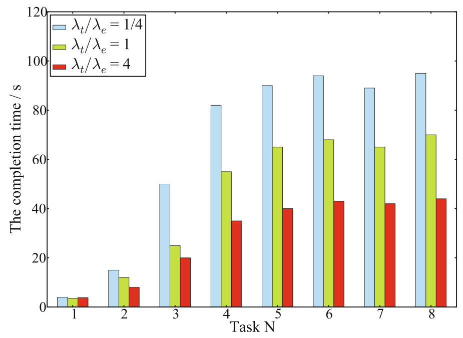


Fig. 4. Impact of wireless channel data transmission rate.

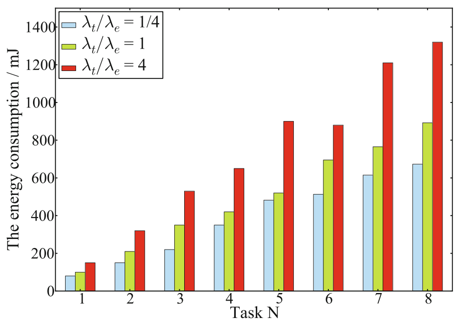
Figure 4 shows the effect of data transmission rate on task execution cost as well as the practicality of our framework. We use the *Face Detection* [1] as experimental application, which needs to identify 99 images and access the cloud via 4G. According to the *Shanon* theorem, we know that the data transmission rate is limited by channel bandwidth and Signal-Noise Ratio(SNR). Therefore, for the wireless access, we set the channel bandwidth $B = 5$ MHz. Theoretically, when the bandwidth B is fixed, the larger the SNR is, the greater the channel data transmission rate is. With the increasing of SNR, the execution cost of the application is decreasing. Compared with other three methods, i.e., all tasks are executed locally, on the central cloud and on the cloudlets, our proposed framework of application partition and optimal cloud selection is much shorter than other methods in term of execution cost. Furthermore, we find that our framework reduces about 80% compared with local execution, and is less about 60% and 65% than the execution in central cloud and cloudlets.

Figure 5 illustrates that the impact of user's preferences, λ_t and λ_e on the completion time and energy consumption of application with different number of tasks. Here, we implement the examination by solving the *N-Queens* problem [8], and give the comparison of execution time and energy consumption for different ratios of $\frac{\lambda_t}{\lambda_e}$. It can be observed from Fig. 5(a) that for a given task, the completion time decreases as λ_t increases, however, the changes of the energy consumption are opposite in Fig. 5(b). This is reasonable since a large λ_t will lead to the little tolerance for completion time for the user. Therefore, the proposed system framework will automatically set the weight value of λ_t to be larger than λ_e, which means that it mainly optimizes the completion time of the application to meet the needs of the user.

Figure 6 demonstrates the comparison of execution cost of our framework with the least context migration system in [9] called framework 1, and the multisite offloading framework using Markov decision process in [13] named framework 2 by the applications of *Face Detection* [1], *N-Queens* [8], *and Sudoku* [2]. We can observe from Fig. 6 that the average overhead in our framework is about 35% less than framework 1, and 30% less than framework 2 in application completion. The reason is that our framework transfers much less data and selects adaptively the optimal cloud.



(a) Completion time.



(b) Energy consumption.

Fig. 5. The impact of user's preference $\frac{\lambda_t}{\lambda_e}$

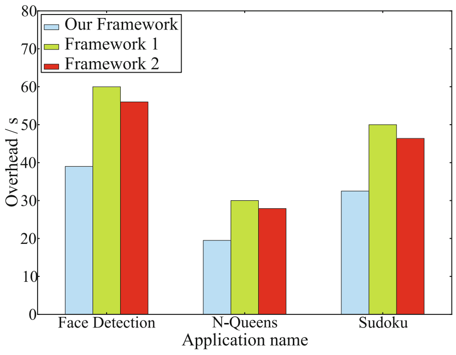


Fig. 6. Comparison of overhead of three framework for different applications.

7 Conclusion

In this paper, we study the execution cost minimization problem in mobile cloud computing. We design an application framework including profiler, solver, and communication module. This framework provides runtime support for adaptive partitioning and dynamic offloading selection in application execution. Under this framework, we propose a novel cloud selection algorithm which is composed of offloading judgement and cloud selection. We implement the framework in a real testbed and experimental results demonstrate that compared to the existing frameworks, our framework can effectively reduce the execution time and energy consumption.

Based on these, the future work will consider the task offloading allocation problem in the multiple network connection ways and variety mobile cloud system scenarios.

Acknowledgement. This work was supported by the National Natural Science Foundation of China (No. 61373178, 61373179, 61402381), Natural Science Key Foundation of Chongqing (cstc2015jcyjBX0094), the Fundamental Research Funds for the Central Universities (XDJK2013A018, XDJK2015C010, XDJK2015D023), and Natural Science Foundation of Chongqing (CSTC2016JCYJA0449), China Postdoctoral Science Foundation (2016M592619) and Chongqing Postdoctoral Science Foundation (XM2016002).

References

1. Face detection.https://facedetection.com/
2. Sudoku. https://play.google.com/store/apps/details?id=com.icenta.sudoku.ui
3. Chabrier, T., Tisserand, A.: On-the-fly multi-base recoding for ECC scalar multiplication without pre-computations. In: 2013 IEEE 21st Symposium on Computer Arithmetic, pp. 219–228 (2013)
4. Chen, X.: Decentralized computation offloading game for mobile cloud computing. IEEE Trans. Parallel Distrib. Syst. **26**, 974–983 (2015)

5. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: CloneCloud: elastic execution between mobile device and cloud. In: Conference on Computer Systems, pp. 301–314 (2011)
6. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: MAUI: making smartphones last longer with code offload. In: International Conference on Mobile Systems, Applications, and Services, pp. 49–62 (2010)
7. Guo, S., Xiao, B., Yang, Y., Yang, Y.: Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016)
8. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: 2012 Proceedings IEEE INFOCOM, pp. 945–953 (2012)
9. Li, Y., Gao, W.: Code offload with least context migration in the mobile cloud. In: 2015 IEEE Conference on Computer Communications (INFOCOM), pp. 1876–1884 (2015)
10. Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., Qureshi, A.: Application partitioning algorithms in mobile cloud computing: taxonomy, review and future directions. J. Netw. Comput. Appl. **48**(C), 99–117 (2015)
11. Khan, A.R., Othman, M., Madani, S.A., Khan, S.U.: A survey of mobile cloud computing application models. IEEE Commun. Surv. Tutor. **16**(1), 393–413 (2014)
12. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
13. Terefe, M.B., Lee, H., Heo, N., Fox, G.C., Oh, S.: Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. Pervasive Mob. Comput. **27**(C), 75–89 (2016)
14. Yang, L., Cao, J., Cheng, H., Ji, Y.: Multi-user computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. Comput. **64**(8), 2253–2266 (2015)
15. Yang, L., Cao, J., Tang, S., Li, T., Chan, A.T.S.: A framework for partitioning and execution of data stream applications in mobile cloud computing. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 794–802 (2012)
16. Yang, S., Kwon, D., Yi, H., Cho, Y., Kwon, Y., Paek, Y.: Techniques to minimize state transfer costs for dynamic execution offloading in mobile cloud computing. IEEE Trans. Mob. Comput. **13**(11), 2648–2660 (2014)