ICIC Express Letters
Part B: Applications
Volume **7**, Number **9**, September **2016**

 ISSN 2185-2766
pp. **1863–1869**

A TIME-EFFICIENT ALGORITHM OF TASK OFFLOADING FOR COMPUTATION-INTENSIVE APPLICATIONS IN MOBILE CLOUD COMPUTING

Ling Li[1], Haiyan Tang[1], Xiuhua Yang[2], Dianshu Chen[1] and Xiaoming Tao[1]

[1]College of Communication Engineering
[2]Network Center of Jilin University
Jilin University
No. 5372, Nanhu Road, Nanhu Campus of Jilin Univeisity, Changchun 130012, P. R. China
{ liling2002; yangxh }@jlu.edu.cn; tanghy14@mails.jlu.edu.cn
{ chendianshu16; taoxiaoming2016 }@sina.com

Received February 2016; accepted May 2016

Abstract. *Recent studies suggest that offloading some components of a computation-intensive application from a mobile device to the cloud for execution is a feasible approach to alleviate the burden of the mobile device. So, a challenging job is to determine which components should be offloaded. In this paper, aiming at minimizing the total application execution time, the Time-efficient Algorithm (TE Algorithm) is proposed. A dynamic programming is adopted firstly, and then computation-intensive tasks are given a priority to be offloaded. Simulation results prove that the Time-efficient Algorithm outperforms the existing optimal solution.*
Keywords: Mobile cloud computing, Offloading decision, Time-efficient Algorithm, Dynamic programming, Computation-intensive task cluster, Preorder task

1. **Introduction.** Nowadays, computation-intensive applications have emerged. However, the processing power of mobile devices is limited [1,2]. One solution is offloading these applications to the cloud for execution [1-4]. The powerful computation capacity of the cloud can eliminate the running time of a mobile application and save energy for the device battery [5-7]. Recently, scholars have been studying on offloading only some parts of the applications [8-10]. In [11], a real-time decision making algorithm suggested fine-grained code offloading. In [12], a code partition algorithm explored the offloading decision on a sequence of calls by a linear time searching scheme. However, these algorithms only aimed at simple tasks, so they could not cover all scenarios. In [13], Jia et al. transformed all the tasks into a general task graph and studied on both sequential tasks and concurrent tasks. However, there was no consideration of reducing the communication cost in his work.

Our main contributions in this paper are as follows. We propose the TE Algorithm for the offloading problems in mobile cloud clouding, and optimally solve the problem of minimizing the total execution time of a mobile application. A dynamic programming algorithm is adopted firstly to look for the boundary offloading tasks from a macroscopic view. In addition, the internal structures of all the complex tasks are studied from a microscopic view. One prominent characteristic of this work is real time adaptability, as the offloading decision is made while the application is at running time. Besides, exploring both simple tasks and complex tasks is a novel idea which is superior to all the previous works, since it covers all aspects of the offloading problems.

The rest of this paper is organized as follows. In Section 2, model formulation of the TE Algorithm is demonstrated. In Section 3, the TE Algorithm is described in detail. In

succession, the TE Algorithm is compared with an existing relatively optimal algorithm in Section 4. Conclusions are drawn in Section 5.

2. **Model Formulation.** For a given mobile application, it is composed of multiple tasks which are executed in time sequence. A task diagram is formed by abstracting each task as a vertex and adopting directed edges to denote the calling relationships. At some moments, there may be only one task being executed, which is described as a simple vertex in a diagram. Whereas, for a cluster of tasks executed simultaneously at a certain time, each cluster can be depicted into a complex vertex.

From a macroscopic view, as Figure 1 shows, a diagram can be converted into a linear structure by expressing both the simple task and the complex task as an 'entity task', where the empty vertices stand for simple tasks, and the filled vertices stand for complex tasks.

From a microscopic view, all complex tasks contain a few simple tasks inside. Figure 2 illustrates an example of the internal structure of a complex task.

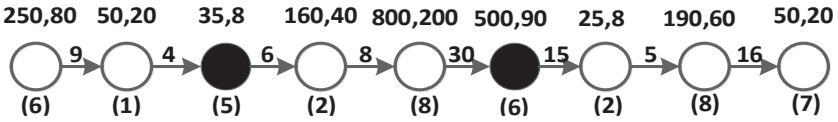

FIGURE 1. Linear structure of a task diagram

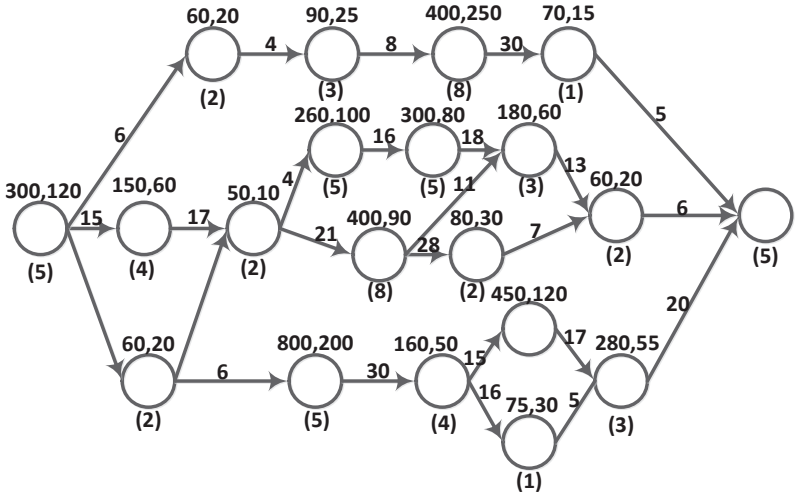

FIGURE 2. Internal structure of a complex task

In a diagram, each task possesses two numerical values: tm and tc.

$$tm_i = \frac{w_i}{pm} \tag{1}$$

$$tc_i = \frac{w_i}{pc} \tag{2}$$

where tm_i and tc_i represent the execution time of $task_i$ on the device and on the cloud respectively, pm and pc stand for the computing power of the mobile device and the cloud respectively, and w_i indicates the average number of instructions of $task_i$.

Assuming $task_i$ and $task_{i+1}$ are two consecutive tasks, and they are executed at different places, thus the data transmission time cost between them can be denoted as:

$$t_{i,i+1} = \frac{h_{i,i+1}}{V} \tag{3}$$

where $h_{i,i+1}$ is the size of data transferring from $task_i$ to $task_{i+1}$, and V stands for the data rate of the wireless communication between the mobile device and the cloud.

If $task_i$ is offloaded, the cost of transferring its program code is:

$$Tcode_i = \frac{Size(code_i)}{V} \tag{4}$$

where $Size(code_i)$ is the size of program code of $task_i$.

In a diagram, the numerical value on each directed edge between two tasks is the time cost of data transferring. The two values marked above each task stand for the tm and tc for this task, respectively. The values in brackets under each task are the time cost of transferring the program code of this task.

3. Time-efficient Algorithm (TE Algorithm).

In this Section, the Time-efficient Algorithm (TE Algorithm) is introduced. Firstly, a dynamic programming algorithm is proposed to find the optimal StartingTask and EndingTask in a linear diagram macroscopically. Then, internal structures of all the complex tasks are further analyzed.

3.1. Macroscopic offloading decision-making.

Existing studies have verified that the transmission cost during application offloading includes two parts: 1) the cost caused by transferring data between two contiguous tasks [9,14]; 2) the transferring cost by migrating the program code of each task. On a linear task diagram, the optimal remote segments must be a series of consecutive tasks without discontinuity and there is only one optimal pair of StartingTask and EndingTask. For example, in Figure 3, it is better to offload $task_4$ together with $task_2$, $task_3$, $task_5$ and $task_6$ than execute $task_4$ locally.

Figure 4 shows a linear diagram that contains K consecutive tasks. We propose a dynamic programming algorithm to find the optimal StartingTask and EndingTask pair, in which all the tasks in the diagram are traversed in order.

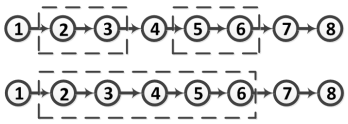

FIGURE 3. Linear structure of a task diagram

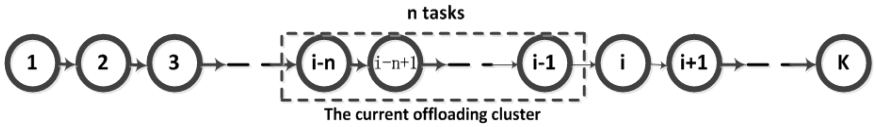

FIGURE 4. Internal structure of a complex task

The offloading decision on $task_i$ is made according to the following step.

Assume that the offloading decisions have already been made on the first $(i-1)$ tasks in the diagram, and the current offloading cluster contains n consecutive tasks which have been decided to be offloaded right before $task_i$. Then, the judging criterion on $task_i$ is shown as Formula (5).

$$TS_i = \begin{cases} TS_{i-1} + inT_i + tm_i - tc_i - Tcode_i, \\ \quad \text{if } (TS_{i-1} + inT_i + tm_i - tc_i) > (tm_i - tc_i - inT_i) \\ tm_i - tc_i - inT_i - outT_i - Tcode_i, \\ \quad \text{otherwise} \end{cases} \qquad (2 \le i \le K) \tag{5}$$

where TS_{i-1} and TS_i denote the current Time Saving after offloading decisions have already been made on $task_{i-1}$ and $task_i$, respectively, and inT_i and $outT_i$ represent the Time Cost of transferring the input data and output data for $task_i$, respectively.

In Formula (5), $(TS_{i-1}+inT_i+tm_i-tc_i-Tcode_i)$ stands for the Time Saving if $task_i$ is decided to be added into the current offloading cluster, $(tm_i-tc_i-inT_i-outT_i-Tcode_i)$ is the Time Saving if $task_i$ is taken to be the new StartingTask. The offloading decision on $task_i$ is made in the circumstance that saves more time. In other words, if the first condition in Formula (5) holds, the offloading decision will be adding $task_i$ into the current offloading cluster. Otherwise, abandon the current offloading cluster and take $task_i$ to be the new StartingTask, and thus a new offloading cluster will be produced. Moreover, TS_i is assigned to values according to the more time-saving condition. After that, the offloading decision will be made on $task_{i+1}$ in this linear diagram in the same way.

Initial values have to be set before the proposed dynamic programming is performed:

$$TS_1 = tm_1 - tc_1 - inT_1 - outT_1 - Tcode_1 \tag{6}$$

Each task is corresponded with a unique Time Saving after the offloading decision has been made on it. After all the tasks have been judged, there will be a certain TS_i with the biggest Time Saving. Then assign the value of the biggest TS_i to $T\max$:

$$T\max = (TS_i)_{\max} \tag{7}$$

where $T\max$ represents the most Time Saving under a specific circumstance, and the StartingTask and EndingTask pair which corresponds with $T\max$ is the optimal pair. That is to say, the optimal offloading decision is to offload all the tasks between the StartingTask and the EndingTask .

Figure 5 shows an inner structure of a complex task. It is worth mentioning that if the macroscopic offloading decision on a complex task (such as the complex task in Figure 5) is to execute it locally, that means its enter task and exit task ($task_1$ and $task_{11}$) will be executed locally; otherwise, its enter task and exit task will be offloaded to the cloud.

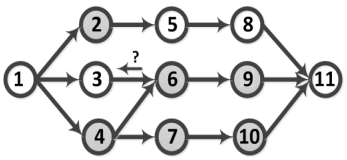

FIGURE 5. Complex task decision-making

3.2. **Microscopic offloading decision-making.** A cluster of consecutive tasks without discontinuity between the StartingTask and the EndingTask has been found in a linear diagram. Then, all the complex tasks are analyzed for further precise decision.

i) Questing for computation-intensive task cluster

A threshold value needs to be set as X, which denotes a dividing point of remote and local segments. For a mobile application, each task corresponds with a 'tm', which represents its execution time on the device. We define the tasks whose tm is larger than X as computation-intensive tasks, which are given priorities to be offloaded.

In Figure 5, $task_2$, $task_4$, $task_9$ and $task_{10}$ are defined to be computation-intensive tasks . Previous works have proved that tasks tend to be offloaded in clusters. Briefly, if a task is offloaded, it is likely that its adjacent tasks are also offloaded. This observation also applies to complex tasks, so all the other tasks that are situated on the connecting lines of these computation-intensive tasks are also selected to join the offloading cluster. Therefore, $task_6$ and $task_7$ are also selected to be offloaded, and a computation-intensive task cluster (including $task_4$, $task_6$, $task_7$, $task_9$ and $task_{10}$) is produced.

ii) Judging on preorder tasks

For a computation-intensive cluster, there will be further decisions making on its pre-order tasks.

Directed Acyclic Graph	Number of Vertices	Number of Edges
DAG 1	14	17
DAG 2	28	39
DAG 3	50	66
DAG 4	100	127

FIGURE 6. DAGs

Assuming that the complex task in Figure 5 has been decided to be offloaded to the cloud for execution in Section 3.1, $task_3$ is the preorder task of the computation-intensive cluster, and the preorder task of $task_3$ is $task_1$. In this case, the decision on $task_3$ is made as the following criterion shows:

$$tm_3 + inT_3 + outT_3 - tc_3 - Tcode_3 > 0 \tag{8}$$

If Formula (8) holds, $task_3$ will be offloaded to the cloud, because executing $task_3$ remotely will save more time than executing it locally. Otherwise, $task_3$ will be executed on the mobile device.

In fact, the complex task in Figure 5 may have been decided to be executed locally in Section 3.1. That is, the preorder task of $task_3$ is executed on the device. Assuming that $task_i$ (such as $task_3$ in Figure 5) is the preorder task of a computation-intensive cluster, and the preorder task of $task_i$ as $task_f$, then the general decision on $task_i$ will be:

a) If $task_f$ has been decided to be offloaded in Section 3.1, the decision will be:

$$\begin{cases} \text{Offloading } task_i, & \text{if } (tm_i + inT_i + outT_i - tc_i - Tcode_i) > 0 \\ \text{Executing } task_i \text{ locally}, & \text{otherwise} \end{cases} \tag{9}$$

b) If $task_f$ has been decided to be executed locally in Section 3.1, the decision will be:

$$\begin{cases} \text{Offloading } task_i, & \text{if } (tm_i - inT_i + outT_i - tc_i - Tcode_i) > 0 \\ \text{Executing } task_i \text{ locally}, & \text{otherwise} \end{cases} \tag{10}$$

4. **Simulation.** In this section, the performance of the TE Algorithm is implemented in Java on MyEclipse 2013 on a dual core Intel Celeron 2.8GHz processor, 4G RAM laptop. The TE Algorithm is compared with two other scenarios: 1) Executing all the tasks locally; 2) Mike Jia's work in [13], which is considered as a relatively optimal algorithm.

We implemented the TE Algorithm on 50 different DAGs (Directed Acyclic Graph). The weights of all the vertices and edges were stochastically valued from a uniform distribution. In detail, the values of w_i were randomly sampled from 150 to 2400. In addition, the processing capacities of the mobile device and the cloud were fixed to $pm = 3$ (instructions per millisecond) and $pc = 15$ respectively. Consequently, according to Formula (1), the values of tm_i range from 50ms to 800ms. As the computation-intensive tasks tended to be time consuming on mobile devices, the threshold value was set to $X = 100$ ms. That is to say, in Section 3.2, we defined all the simple tasks of which tm_i were larger than 100ms as computation-intensive tasks in our experiment. The network bandwidth was considered as a variable and it varied from 0.1 to 2 for observation. For simplicity, we randomly selected four DAGs (as shown in Figure 6) to present in the paper.

From Figure 7, it can be apparently found that executing the entire application locally turned out to be the most time consuming of all the scenarios. Moreover, the TE Algorithm outperformed Mike Jia's algorithm on the whole. 1) In TE Algorithm, the computing capacity of the mobile device was weighed firstly and a threshold value was set as the boundary standard of distinguishing whether a task was supposed to be offloaded. These approaches not only enhanced execution effects with the aid of the cloud's strong

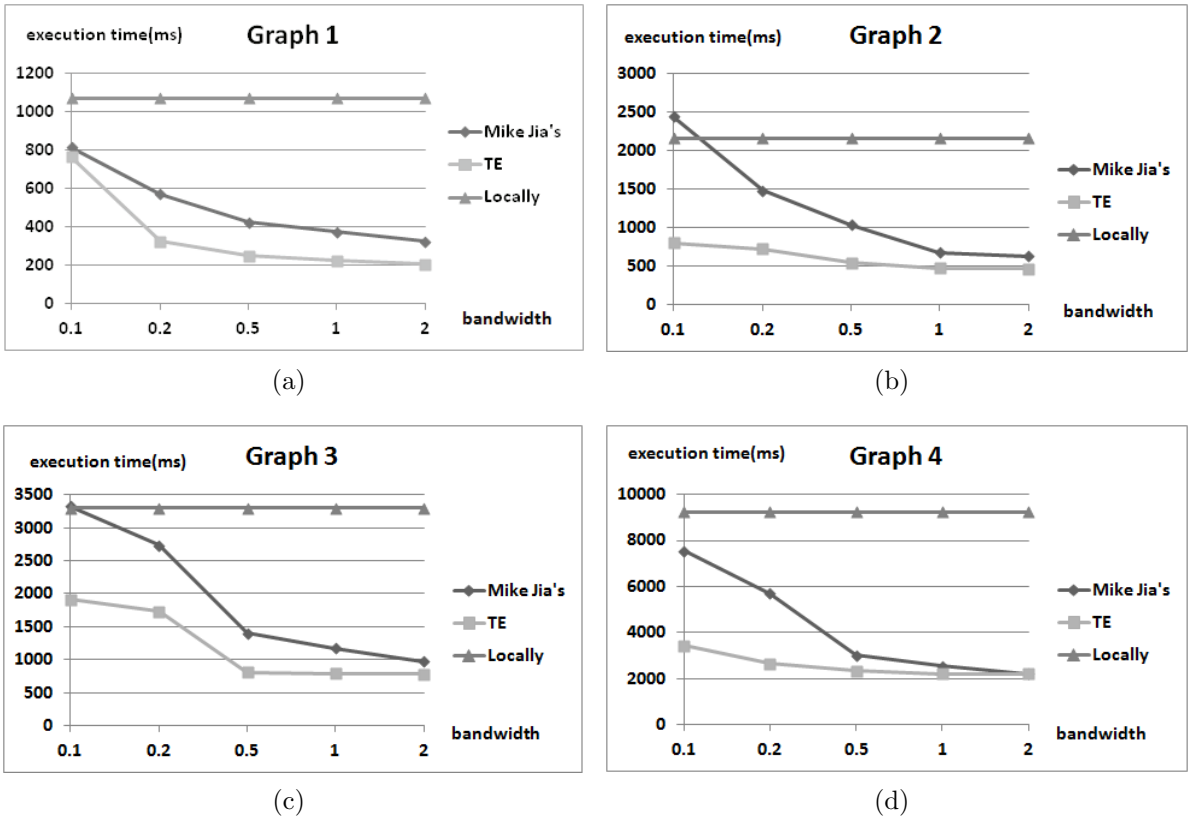

FIGURE 7. Simulation results

computing power, but also avoided resources waste caused by device idle. 2) In TE Algorithm, searching for computation-intensive task clusters and judging on the preorder tasks contributed to a lower-level communication cost when transferring data. 3) For concurrent tasks, Mike Jia's load-balancing scheme did not have attempts to decrease the data transmission time.

5. **Conclusions.** In this paper, the Time-efficient Algorithm is proposed to minimize the total execution time for computation intensive applications in mobile cloud computing. A dynamic programming algorithm is firstly conducted to determine the optimal StartingTask and EndingTask in a linear task diagram. Then, computation-intensive tasks are given priorities to be offloaded and the preorder tasks are judged. Evaluation has shown that the Time-efficient Algorithm outperforms the approximately optimal algorithm in [13]. In this case, Time-efficient Algorithm can be applied to mobile applications, which is bound to lessen battery energy. For future direction, we will explore the effectiveness of TE Algorithm in terms of energy saving, which provides far-reaching significance on extending battery life.

Acknowledgment. This work is partially supported by foundation project for Science and Technology Department of Jilin Province (Grant No. 20150204008GX) and foundation project for the Education Department of Jilin Province (Grant No. 2014B006). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] N. Fernando, S. W. Loke and W. Rahayu, Mobile cloud computing: A survey, *Future Generation Computer Systems – The International Journal of Grid Computing and Escience*, vol.29, pp.84-106, 2013.

[2] O. S. Mao, Y. Kun and L. Antonio, Performance analysis of offloading systems in mobile wireless environments, *IEEE International Conference on Communications*, Glasgow, Scotland, pp.1821-1826, 2007.

[3] G. Mohanarajah and D. Hunziker, Rapyuta: A cloud robotics platform, *IEEE Trans. Automation Science and Engineering*, vol.12, pp.481-493, 2015.

[4] M. Satyanarayanan, P. Bahl and R. Caceres, The case for VM-based cloudlets in mobile computing, *IEEE Pervasive Computing*, vol.8, pp.14-23, 2009.

[5] C. A. Ardagna and C. Mauro, An anonymous end-to-end communication protocol for mobile cloud environments, *IEEE Trans. Services Computing*, vol.7, pp.373-386, 2014.

[6] M. Choi and J. Park, Mobile cloud computing framework for pervasive and ubiquitous environment, *Journal of Supercomputing*, vol.64, pp.331-356, 2013.

[7] D. S. Guang, H. L. Tao and T. Javid, Computation offloading for service workflow in mobile cloud computing, *IEEE Trans. Parallel and Distributed Systems*, vol.26, pp.3317-3329, 2015.

[8] B. G. Chun and P. Maniatis, Dynamically partitioning applications between weak devices and clouds, *Proc. of the 1st ACM Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond*, no.7, pp.27-34, 2010.

[9] Z. Yang, N. Dusit and W. Ping, Offloading in mobile cloudlet systems with intermittent connectivity, *IEEE Trans. Mobile Computing*, vol.14, pp.2516-2529, 2015.

[10] M. Shiraz and A. Gani, A lightweight active service migration framework for computational offloading in mobile cloud computing, *Journal of Supercomputing*, vol.68, pp.978-995, 2014.

[11] E. Cuervoy, A. Balasubramanian and D.-K. Cho, MAUI: Making Smartphones last longer with code offload, *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, New York, pp.49-62, 2010.

[12] Y. Zhang, H. Liu, L. Jiao and X. Fu, To offload or not to offload: An efficient code partition algorithm for mobile cloud computing, *IEEE the 1st International Conference on Cloud Networking*, 2012.

[13] M. Jia, J. Cao and L. Yang, Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing, *IEEE Conference on Computer Communications*, Toronto, pp.352-357, 2014.

[14] L. Bo, P. Y. Jian and W. Hao, Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds, *Journal of Supercomputing*, vol.71, pp.3009-3036, 2015.