Effective Computation Offloading Schemes via Application Partitioning and VM Allocation in Mobile Cloud Environment

Heejae Kim and Chan-Hyun Youn

Dept. of Electrical Engineering
Korea Advanced Institute of Science and Technology (KAIST), Korea
{kim881019,chyoun}@kaist.ac.kr

Abstract. Mobile devices are rapidly developing and the mobile applications are being complicated correspondingly. For seamless executions of the complex applications, the obstacles in mobile devices such as hardware and battery limitation should be overcomed. In this paper, we present effective compuatation offloading schemes via application partitioning and VM allocation in mobile cloud environment. For application partitioning, heuristics for offloading method decision in the mobile cloud (OMD-MC) are presented and it is operated energy-efficiently with low computational complexity. Also, VM allocation for the methods to be offloaded is also discussed for the cost reduction in mobile cloud. We first formulate an initial VM allocation to minimize the total computation and transmission cost in a cloud and describe how to improve it via method duplication. Evaluation results show that the OMD-MC is operated energy-efficiently compared with other two approaches and the method duplication is effective to reduce the total cost in the cloud.

Keywords: Computation offloading, application partitioning, VM allocation.

1 Introduction

With rapid development of mobile devices, mobile applications are being more complex. As the applications to be bigger, the mobile devices should be capable for seamless executions. However, current mobile devices have three obstacles to handle the complex applications: limitation of hardware and battery, mobility of mobile devices, and security problem [1].

Computation offloading is an effective solution to overcome the obstacles by moving computation to other machines which have more resource [2]. Especially, in mobile cloud environment, the offloading is operated from a mobile device to a cloud. One of the challenges in the offloading is deciding the portion of an application to be offloaded. Because some parts of the application are not beneficial if they are offloaded, and computation in a cloud and data transmission depends on the real-time node and network status respectively, the application should be partitioned effectively. To address the problem, many researches presented schemes for energy saving [3, 4, 5, 6, 7] or performance improving [8, 9, 10] in the offloading [2]. The goals are to

handle battery and hardware limitation issues in mobile devices respectively. Also, the both goals are not independent and achievements of each goal makes a positive effect to the other goal. In this paper, we present heuristics for offloading method decision in a mobile cloud (OMD-MC) to achieve effective application partitioning. For an application which has independent methods, the OMD-MC is operated using the direct acyclic graph representing execution flows (EF-DAG) of the application and it is designed to decide methods in the EF-DAG to be offloaded to the cloud energy-efficiently with low complexity.

Because cloud computing essentially follows pay-as-you-go model, the cost reduction is an important issue in the computation offloading in mobile cloud environment, we also discuss VM allocation for the methods to be offloaded. In the VM allocation, an initial VM allocation problem is firstly formulated to achieve maximal cost-effectiveness using the direct acyclic graph representing execution flows in the cloud (CEF-DAG) and a scheme to improve the initial VM allocation via method duplication is also described.

The remainder of this paper is organized as follows. Section 2 presents the EF-DAG and OMD-MC for the application partitioning for computation offloading. Section 3 discuss VM allocation for the methods in the CEF-DAG, and present a formulation for initial VM allocation and its improvement. In section 4, we evaluate the OMD-MC and the VM allocation scheme. Finally section 5 concludes this paper.

2 Application Partitioning for Computation Offloading

2.1 EF-DAG

We handle applications which are composed of independent methods. Fig. 1 shows an example of dependencies between the methods in the application. In the figure, two vertices each connected are regarded as the two methods having dependency and it means that the two methods affects each other during the overall execution of the application (e.g., if the result of one method becomes the input of the other method).

To specify the dependency, we present EF-DAG which is the direct acyclic graph representing execution flows of the applications and Fig. 2 shows an example of the EF-DAG using the application in Fig. 1. In the EF-DAG, each vertex $v \in V$ represents the method where V is the set of the methods in the application. The weight of a vertex v represents the energy consumption for the computation of the vertex and is represented as $\varepsilon(v)$. If the vertex v is used for several times during the overall execution, we denote it as the vertex v' for the second usage, the vertex v'' for the third usage, and so on. We note that the vertices colored in gray represent unoffloadable vertices. We denote the weight of an edge $e_{ij} \in E$ as $\varepsilon(e_{ij})$ and it represents the energy consumption for data transmission from vertex i to j where the E is the set of the edges assuming that the vertex i is executed in the mobile device and the vertex j is executed in the cloud.

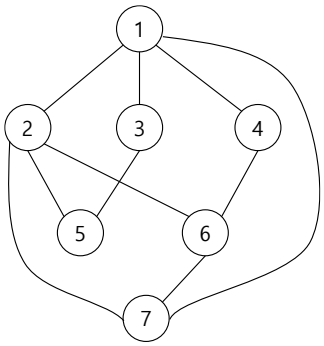


Fig. 1. An example of dependencies between the methods in the application

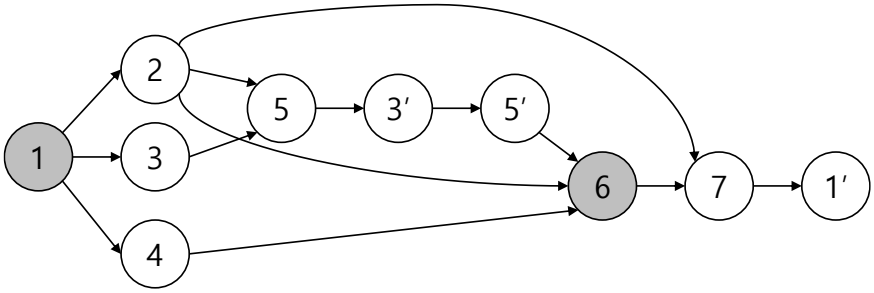


Fig. 2. The EF-DAG using the application using the application in Fig. 1.

2.2 OMC-MC

In this section, we present the OMD-MC to decide which vertices in the EF-DAG are to be offloaded to the cloud. The goal of the OMD-MC is to reduce energy consumption in the mobile device with guaranteeing the overall execution of the application finishes before the deadline similarly with Cuervo et al. [3]. Algorithm 1 shows the OMD-MC and it consists of two steps: initial setting and vertex selection.

Initial setting. In initial setting, unoffloadable methods are classified and the set S^m is generated as $V - \{\text{unoffloadable vertices}\}$. $T^{m,comp}$ is initially defined as the sum of computation times of the unoffloadable vertices in the mobile device.

Vertex selection. In vertex selection, the vertex v^* to execute in the mobile device in the set S is decided. In this step, energy consumption changes (ECCs) for every vertex $v \in S$ in the mobile device are calculated first. Eq. (1) depicts the ECC when a vertex $v \in S$ as well as vertices in $V - S$ is not offloaded.

$$\Delta ECC^{m}_{(V-S)\cup\{v\in S\}} = \varepsilon(v) + \left(\sum_{j\in S}\varepsilon\left(e_{vj}\right) + \sum_{i\in S}\varepsilon\left(e_{iv}\right)\right) - \left(\sum_{j\in V-S}\varepsilon\left(e_{vj}\right) + \sum_{i\in V-S}\varepsilon\left(e_{iv}\right)\right). \quad \textbf{(1)}$$

Then, the vertex v^* is decided to minimize the ECC with the deadline constraint given in Eq. (2) where $t^{m,comp}_{v^*}$ is the computation time of the vertex v^* in the mobile device, CP^m is the critical path of the EF-DAG, $E\left[t^{c,comp}_{v}\right]$ is the expected completion time of the vertex v in the cloud, $t^{tran}_{e_{ij}}$ is the transmission time from the vertex i to j, and $u(i,j)$ is a variable whose value is 1 if there occurs transmission between the vertex i and j, and 0 otherwise.

$$T^{m,comp} + t^{m,comp}_{v^*} + \sum_{v\in CP^m\cap S} E\left[t^{c,comp}_{v}\right] + \sum_{i,j\in CP^m} t^{tran}_{e_{ij}}\cdot u(i,j) \le deadline \quad \textbf{(2)}$$

Finally, the vertex v^* is removed from the set S, and $T^{m,comp}$ is updated. The vertex selection is repeated until the set S is empty and the deadline constraint is violated.

Algorithm 1. OMD-MC

1: $T^{m,comp} \leftarrow \sum_{v\in V} t^{m,comp}_{v}$, $\forall$unoffloadable vertices

2: $S \leftarrow V - \{\text{unoffloadable vertices}\}$

3: while $S \neq \varnothing$

4: for $v \in S$

5: if $T^{m,comp} + t^{m,comp}_{v^*} + \sum_{v\in CP^m\cap S} E\left[t^{c,comp}_{v}\right] + \sum_{i,j\in CP^m} t^{tran}_{e_{ij}}\cdot u(i,j) > deadline$

6: break

7: end if

8: end for

9: for $v \in S$

10: $\Delta ECC^{m}_{(V-S)\cup\{v\in S\}} = \varepsilon(v) + \left(\sum_{j\in S}\varepsilon\left(e_{vj}\right) + \sum_{i\in S}\varepsilon\left(e_{iv}\right)\right) - \left(\sum_{j\in V-S}\varepsilon\left(e_{vj}\right) + \sum_{i\in V-S}\varepsilon\left(e_{iv}\right)\right).$

11: end for

12: select v^* which minimizes $\Delta ECC^{m}_{(V-S)\cup\{v\in S\}} < 0$ and $T^{m,comp} + t^{m,comp}_{v^*} + \sum_{v\in CP^m\cap S} E\left[t^{c,comp}_{v}\right] + \sum_{i,j\in CP^m} t^{tran}_{e_{ij}}\cdot u(i,j) \le deadline$

13: remove v^* from S

14: $T^{m,comp} = T^{m,comp} + t^{m,comp}_{v^*}$

15: end while

3 VM Allocation for Methods in CEF-DAG

3.1 Formulation for the Initial VM Allocation

As the result of the OMD-MC, the elements in the final set S are decided to be executed in the cloud. For VM allocation for those methods, we present CEF-DAG. The CEF-DAG is the same as the EF-DAG except that the vertices to be executed in the mobile device are excluded and the vertex set is the final set S. Therefore, we denote the corresponding edge set as E^c. We note that edges from the set $V-S$ to S or from the set S to $V-S$ in the E are remained in the E^c. These edges are denoted as e_{0v} or e_{0v} for every vertex $v \in S$. The weight of a vertex $v \in S$ represents the computation cost in VM type k, and it is denoted as $c\left(v,k\left(v\right)\right)$ where $k\left(v\right)$ is the VM type in which the vertex v is allocated. Also, the weight of an edge $e_{ij} \in E^c$ represents the transmission cost from the vertex i to j, and it is denoted as $c\left(e_{ij}\right)$.

We formulate an optimization problem for the initial VM allocation. The objective of the problem is to find $c\left(v,k\left(v\right)\right)$ and $c\left(e_{ij}\right)$ for every $v \in S$ and $e_{ij} \in E^c$ respectively satisfying Eq. (3) whose objective function is the total cost in the cloud (TC^c) with constraints of Eq. (4). Eq. (4) represents the deadline constraint where $t_{v,k(v)}^{c,comp}$ is completion time of the vertex v in VM type $k\left(v\right)$ and CP^c is the critical path of the CEF-DAG.

$$\text{minimize } TC^c = \sum_{v \in V^c} c\left(v,k\left(v\right)\right) + \sum_{e \in E^c} c\left(e_{ij}\right) \tag{3}$$

$$\text{subject to } \sum_{v \in CP^c} t_{v,k(t)}^{c,comp} + \sum_{i,j \in CP^c} t_{e_{ij}}^{tran} \le deadline \tag{4}$$

3.2 Improvement of the Initial VM Allocation via Method Duplication

In this section, we present a scheme to improve the initial VM allocation via method duplication. The basic idea of the scheme is that usage of new VMs to duplicate methods can be more cost-effective if the computation costs in the new VMs are less than the transmission costs between methods. The concept of method duplication can be applied for the dynamic case when there occurs performance variation in the cloud.

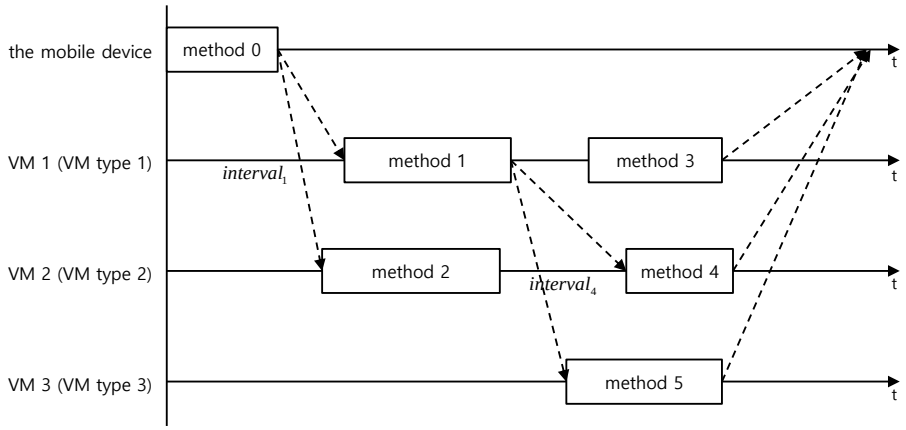
The scheme is shown in Algorithm 2 (Line 2 ~ Line 8). After the initial VM allocation, $interval_j$ is firstly calculated for each vertex $j \in S$. The $interval_j$ of a vertex j denotes the time interval in which no method is allocated before the

computation of the vertex j. For every $interval_j$, if there are vertex $i \in S$ satisfying that the computation time of the vertex i in the VM type $k(j)$ is less than $interval_j$, and the transmission cost of e_{ij} is greater than the computation cost of the vertex i in the VM type $k(j)$, new VMs are created for the computation of the vertex i in the $interval_j$.

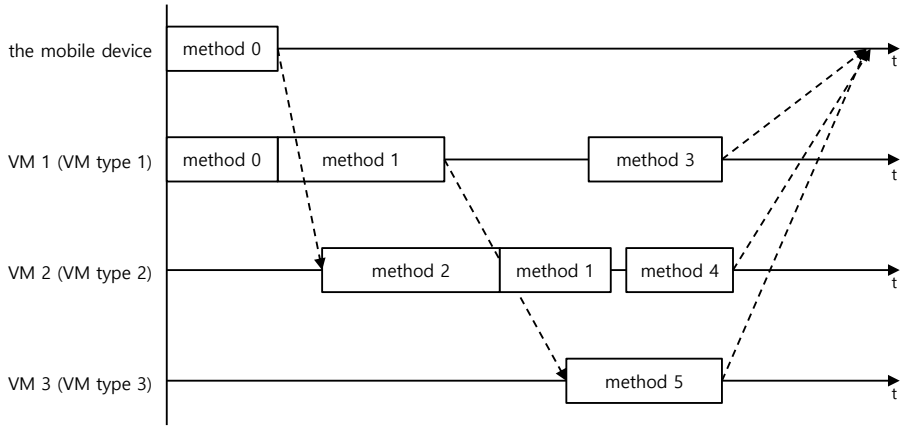
Fig. 3(b) is an example of the improvement of the initial VM allocation as shown in Fig. 3(a). Because $t^{c,comp}_{0,k(1)}$ is less than $interval_1$ and $c(e_{01})$ is greater than $c(0,k(1))$, the method 0 is duplicated in VM 1. Also, because $t^{c,comp}_{1,k(2)}$ is less than $interval_4$ and $c(e_{14})$ is greater than $c(1,k(2))$, the method 1 is duplicated in VM 2. Therefore, new VMs are created for the computation of the method 0 and 1 before that of the method 1 and the method 4 respectively.

Algorithm 2. VM allocation for methods of the CEF-DAG in the cloud

1: initial allocation
2: for $interval_j, \forall j \in S$
3: for $i \in S$
4: if $t^{c,comp}_{i,k(j)} < interval_j \wedge c(e_{ij}) > c(i,k(j)), \forall e_{ij} \in E^c$
5: a new VM is created for the computation of the vertex i in the $interval_j$
6: end if
7: end for
8: end for



(a)



(b)

Fig. 3. Improvement of the initial VM allocation via method duplication: (a) before the improvement. (b) after the improvement.

4 Evaluation

We evaluate the OMC-MC and the scheme for VM allocation in this section. For the evaluation, we consider the application which has the EF-DAG as shown in Fig. 4. In

the figure, vertex and edge weights are presented, and the values in the brackets denotes the transmission time between two vertices.

The OMC-MC is evaluated in two cases: with and without the deadline which is 46. In addition, we compare the OMC-MC with two other approaches: computing all vertices in the mobile device and computing all offloadable vertices in the cloud. The evaluation results of the OMC-MC are shown in Fig. 5. The energy consumption is the smallest using the OMC-MC without the deadline. With the deadline, the energy consumption increases and it means the transmissions between the mobile device and the cloud as well as the computations highly affect the execution time of the application. With the same reason, the result shows that the difference of the energy consumption between computing all vertices in the mobile device and all offloadable vertices in the cloud is relatively small.

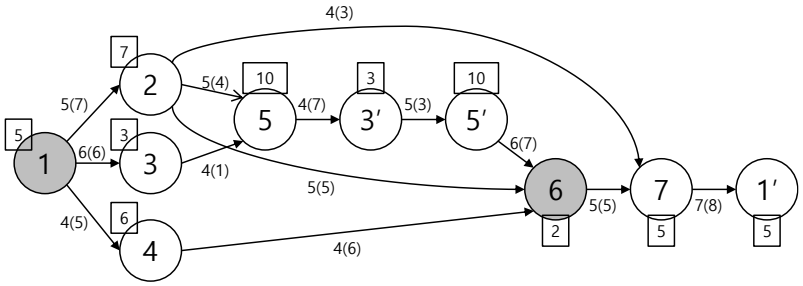


Fig. 4. The EF-DAG for the evaluation.

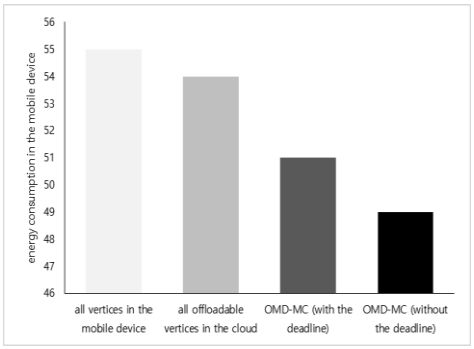


Fig. 5. Energy consumptions in the mobile device when all vertices being executed in the mobile devices, all offloadable vertices being executed in the cloud, and using the OMD-MC with and without the deadline.

The CEF-DAG which is constructed as the result of the OMC-MC without deadline is depicted in Fig. 6. Using the CEF-DAG, we evaluate the VM allocation scheme. In the evaluation, we assume that VMs are homogeneous, and the

computation and transmission cost are proportional to the computation and transmission time respectively. Therefore, the computation and the transmission time are supposed to be equal to the computation and the transmission cost as setting that the unit computation and the unit transmission cost are 1. The VM allocation scheme is evaluated in two cases: with and without the improvement via method duplication. The evaluation results of the VM allocation scheme are shown in Fig. 7. The total cost in the cloud with the improvement is less than that without the improvement. Also, if we use the applications which is highly parallelized and have large methods, the difference of total cost between the two cases is much larger.

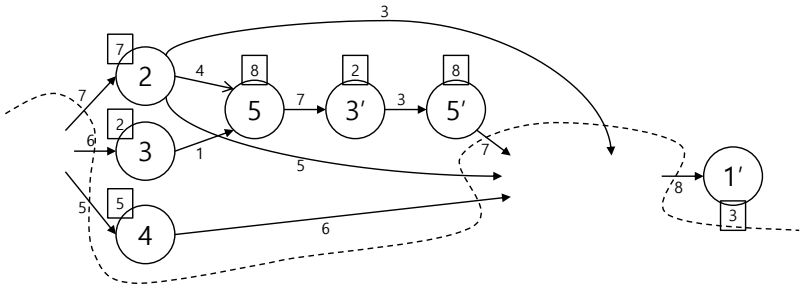


Fig. 6. The CEF-DAG constructed as the result of the OMD-MC.

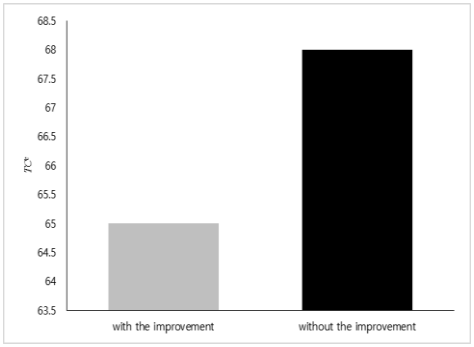


Fig. 7. The total cost in the cloud with and without the improvement of the initial VM allocation via method duplication.

5 Conclusion

In this paper, we presented effective computation offloading schemes via application partitioning and VM allocation in mobile cloud environment. To overcome the

obstacles in mobile devices, the OMD-MC is presented to reduce energy consumption in the mobile device with the deadline constraint and operated using the EF-DAG. In addition, the initial VM allocation for the methods in the CEF-DAG are formulated and the scheme for the improvement of it via method duplication is also discussed to achieve cost-effective VM allocation. The evaluation results showed that the energy consumption in the mobile device using the OMD-MC is smaller than other two approaches and the total cost in the cloud decreases after the improvement is applied. As on-going and future work, we are extending the OMD-MC to be applied in the dynamic environment by adaptively revising the application partitioning and developing heuristics for the initial VM allocation to reduce computational complexity.

Acknowledgments. This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2014, and the MSIP under the ITRC (Information Technology Research Center) support program (NIPA-2014(H0301-14-1020)) supervised by the NIPA (National IT Industry Promotion Agency).

References

1. Satyanarayanan, M.: Fundamental Challenges in Mobile Computing. In: 15th Annual ACM Symposium on Principles of Distributed Computing, pp. 1--7 (1996)
2. Kumar, K., Liu, J., Lu, Y. -H., Bhargava, B.: A Survey of Computation Offloading for Mobile Systems. In: Mobile Network and Applications, vol. 18, no. 1, pp. 129--140 (2013)
3. Cuervo, E., Balasubramanian, A., Cho, D. -K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: MAUI: Making Smartphones Last Longer with Code Offload. In: 8th International Conference on Mobile Systems, Applications, and Services, pp. 49--62 (2010)
4. Kumar, K., Lu, Y. -H: Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?. In: IEEE Computer, vol. 43, no. 4, pp. 51--56 (2010)
5. Ge, Y., Zhang, Y., Qiu, Q., Lu, Y. -H.: A Game Theoretic Resource Allocation for Overall Energy Minimization in Mobile Cloud Computing System. In: 18th ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 279--284 (2012)
6. Wen, Y., Zhang, W., Luo, H.: Energy-optimal Mobile Application Execution: Taming Resource-poor Mobile Devices with Cloud Clones. In: 31st Auunal IEEE International Conference on Computer Communications, pp. 2716--2720 (2012)
7. Huang, D., Wang, P., Niyato, D.: A Dynamic Offloading Algorithm for Mobile Computing. In: IEEE Transactions on Wireless Communications, vol. 11, no. 6, pp. 1991--1995 (2012)
8. Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., Milojicic, D.: Adaptive Offloading for Pervasive Computing. In: IEEE Pervasive Computing, vol. 3, no. 3, pp. 66--73 (2004)
9. Chu, H. H., Song, H., Wong, C., Kurakake, S., Katagiri. M.: Roam, a Seamless Application Framework. In: Journal of Systems and Software, vol. 69, no. 3, pp. 209--226 (2004)
10. Ou, S., Yang, K., Liotta, A.: An Adaptive Multi-constraint Partitioning Algorithm for Offloading in Pervasive Systems. In: 4th Annual IEEE International Conference on Pervasive Computing and Communications, pp. 116--125 (2006)