International Journal of Control and Automation
Vol. 10, No. 10 (2017), pp.163-174
http//dx.doi.org/10.14257/ijca.2017.10.10.14

Predictive Offloading Method and Framework Design Model for Efficient Mobile Cloud Computing

Zhen Zhe Piao[1] and Soo Dong Kim[2,*]

[1,2] *Dept of Computer Science, Soongsil University, Korea,*
[1]*jincheul826@gmail.com,* [2]*sdkim777@gmail.com*

Abstract

With the emergency of the mobile computing, more and more useful mobile applications make it possible for a user to manage personal schedule, entertainment, and do many useful activities. Offloading is a common method to remedy the resource constraint problem of mobile devices. That is, resource-heavy and computation-intensive components are offloaded to a resource-rich server, which runs the components and returns the result. The conventional dynamic offloading approach is to migrate computation-intensive offloadable components after offloading situation occurrence. To resolve the mentioned problem, we propose a predictive offloading framework which migrates computation-intensive offloadable components in advance. As the result, the limitations of the conventional dynamic offloading are overcome, the overall efficiency of offloading is greatly increased. In this paper, we will present the predictive offloading method for efficient mobile cloud computing. At last, we will present experiment result for validating applicability and practicability of our proposal.

Keywords*: Mobile Computing, Mobile Cloud Computing, Component Offloading, Predictive Component Offloading*

1. Introduction

Mobile cloud computing is a computing paradigm that extends cloud computing capabilities to mobile computing [1][2]. Nowadays, mobile computing has become a common computing paradigm that provides convenience to people's daily life. More and more useful mobile applications make it possible for a user to manage personal schedule, entertainment, and do many useful activities. However, some approaches indicate that there is an inherent defect that is the performance of mobile device in comparison to traditional computer that a mobile device is constrained by processor speed, memory size, and *etc* [3]. As a remedy scheme to solve the problem, component offloading makes room to handle mentioned issues via migrating computationally intensive component to the remote server [4]. Due to the flexible access to the internet through a wireless network, we now have the chance to make a mobile device available to interact with cloud nodes.

In this paper, we propose a predictive offloading framework for efficient offloading. Our strategy to increasing the efficiency of mobile cloud applications is to utilize a look-ahead method to identify the specific time point for performing component offloading. We propose an efficient algorithm to predict the demand for resources in advance. Based on the prediction made, our proposal performs component offloading and/or migration in background. In this way, we can eliminate the time to analyze the resource demands at runtime and the time to migrate computationally intensive components. The key benefit of applying prediction is the increasing performance of mobile cloud computing.

* Corresponding Author

ISSN: 2005-4297 IJCA
Copyright © 2017 SERSC Australia

2. Related Works

With the increasing penetration of mobile phone in the world, more people are managing their daily activities using various useful mobile applications. However, some approaches indicate that compared to traditional computer the performance of mobile device is constrained by processor speed, memory size, and *etc* [5][6][7].

In order to resolve the mentioned issues in mobile computing, Abolfazli's work presents a mobile augmentation model which is Cloud-based Mobile Augmentation (CMA). The CMA employs resource-rich clouds to increase, enhance, and optimize computing capabilities of mobile devices for executing resource-intensive mobile applications [8]. Fernando's work specifies a Mobile Cloud Computing paradigm that can execute mobile applications on external resource providers that provide computational capabilities to enhance the performance of a mobile device by exploiting external resource providers in the cloud [9]. Both of the work indicate that the resource-rich cloud computing provides the chance to execute some computationally intensive components of an application on the cloud server for saving energy and improve the performance of a mobile device. La's work proposes a comprehensive and practical self-stabilizing process and its management-related methods for resolving limited computing power and resource constraint problems [10]. The work clearly specifies key elements of the mobile cloud computing environment and their relationships for leveraging mobile cloud computing capability. Moreover, this work presents 4 kinds of remedy actions that include component offloading for improving application quality of a mobile device. Some current works exploit component offloading technique to achieve energy savings and increasing computational capabilities [11][12][13]. For leveraging capability of the component offloading, in this work, we will propose a predictive offloading framework and its design model. In this paper, we will propose two algorithms for achieving the efficient offloading goal.

In Wu's work, a distributed mobile cloud service model POEM manages mobile cloud resources [14]. POEM is able to manage resources not only between mobile devices and cloud servers, but also among mobile applications. The POEM manager performs component offloading, mobile device discovery, and other resource management tasks. Cuervo's work proposes a Mobile Assistance Using Infrastructure (MAUI) system to achieve energy savings under the connection constraints of the mobile device [15]. The MAUI enables developers to distinguish local and remote methods of an application by adding simple annotations to methods at design stage. A profiler of the MAUI makes decisions on remote methods identification and assesses resource usage of methods. Moreover, a solver of the MAUI determines the remote destination of an identified remote method under the profiler component supports. Chun's work proposes Clone Cloud system migrates a thread to a device-level clone in a computational cloud node and reintegrates the thread to the local mobile device [16]. The system performs static analysis to distinguish offloadable codes to identify both migration and reintegration places in the code. The Clone Cloud constructs a cost model on energy consumption of a mobile device via using a dynamic profiler component collected data. Some other similar works present component offloading mechanisms to improve the quality of mobile applications [17][18][19]. The current component offloading technique contains three phases which include computationally intensive component recognition, component migration, and component execution. Moreover, the current offloading approaches are aiming to perform immediate offloading actions after recognizing computationally intensive component without estimating the trend of resource status in near future time point. The proposed predictive offloading framework tracks real-time resource status of client mobile and conducts efficient offloading strategy. The framework analyzes recent historical resource status data and predicts the resource utilization trend for efficient offloading.

For component offloading, some approaches propose efficient offloading algorithms

 Copyright © 2017 SERSC Australia

and methods. Li's work proposes a mobility prediction based computational offloading [20]. The proposal of the work is to select the surrogate node proactively and predict the mobility of client mobile nodes before making the decision to offload. Based on that, future topology of client nodes and availability of surrogate nodes may be predictable. The limitation of the mentioned work is to meet the minimum execution time and minimum completion time when performing component offloading. We proposed a predictive algorithm which makes offloading decision by tracking real-time resource utilization of a client mobile device. Some proposed methods are considering energy consumption of a mobile device for efficient component offloading [21][22]. In our work, the framework manages a number of available offloading service nodes for component offloading. For efficient data transmission over the wireless network, our proposal also considers transmitted data size like Yang's work [23] and extra software module installation. Moreover, in order to eliminate the node selection workload for a client mobile node, the framework selects one or more available offloading service nodes from the separate dispatcher and learner node.

3. Algorithm for Prediction Offloading Component

Prediction is the key strategy of our offloading framework which maximizes the efficiency in ever-changing system environment. The proposed framework determines component offloading based on real-time resource utilization.

A mobile app consists different types of components, and some of them are offloadable. When a client mobile node meets a certain condition, the framework determines to offload candidate components to the offloading service node. For clearly specifying proposed prediction algorithm, we define terms and explanations in Table 1.

Table 1. Using Terms for Prediction Algorithm Explanation

Term	Description
CMN_n	A client mobile node n
$COMP_n$	An offloadable component of an application
$F_{CPU}(CMN_n)$	A threshold value of CPU utilization of a CMN_n
$F_{CPU}(COMP_n)$	A threshold value of CPU utilization for running a $COMP_n$ in a CMN_n
$G_{CPU}(CMN_n, T_i)$	A runtime CPU utilization value of a CMN_n at time T_i
$G_{AVG\_CPU}(CMN_n)$	A recent average CPU utilization value of a CMN_n
$H_{CPU}(CMN_n, T_i)$	A predicted value of CPU utilization of a CMN_n at time T_i
Δd	A deviation value between $G_{CPU}(CMN_n, T_i)$ and $H_{CPU}(CMN_n, T_i)$

Between the CPU utilization and the battery there exists a linearity relationship. The proposed framework tracks component CPU utilization at runtime and makes decision on when to perform component offloading. The framework analyzes recent CPU utilization trend and estimates the future utilization value at a specific time T_i. After that, the framework compares the predicted value with a threshold value of CPU utilization and determines the component offloading starting time. Our approach is to compare predicted CPU utilization value with the threshold value and apply the moving average (MA) for making the decision about the component offloading timing as shown in

Copyright © 2017 SERSC Australia

Table 2.

 Copyright © 2017 SERSC Australia

Table 2. Prediction Algorithm for Determining Offloading Timing

Algorithm#1: Prediction Algorithm for Offloading Timing

Input: $F_{CPU}(CMN_n)$, $G_{CPU}(CMN_n, t)$
Output: Predicted Time T_i

```
1:  if G_AVG_CPU(CMN_n) > F_CPU(CMN_n)
2:     for all I number of predictions:
3:        H_CPU(CMN_n, T_i) = Σ_{i=1} G_CPU(CMN_n, T_{i-1})/n
4:        Δd = H_CPU(CMN_n, T_i) – G_CPU(CMN_n, T_i)
5:        if H_CPU(CMN_n, T_i) >= F_CPU(CMN_n) && Δd>0:
6:           To begin component offloading from the predicted
             time T_i
7:        end if
8:     end for
9:  end if
10: return T_i
```

The monitoring scheme is to calculate average CPU utilization with a specified number of monitoring times and the time interval between monitoring events is same. When the calculated average value $G_{AVG\_CPU}(CMN_n)$ is higher than the defined threshold value $F_{CPU}(CMN_n)$, the framework begins to predict future CPU utilization at time T_i. In this paper, we applied moving average method to predict the CPU utilization value at future time point T_i that is presented from line 2 to line 8. The moving average is a widely used trend detection method that is suitable for making a reliable offloading decision with given recent series data for resource utilization trend determination [24][25][26].

In order to offload more precisely, the algorithm compares some predicted CPU utilization values with runtime CPU utilization values. If all of the compared deviation values between predicted values and real-time values are less than predefined deviation value Δd, the framework determines the time T_i as the offloading timing for performing component offloading.

Table 3. Dynamic Component Selection Algorithm

Algorithm#2: Dynamic Component Selection Algorithm for Component Offloading

Input: $COMP_n$
Output: $OFFLD_{CMN}(COMP_n)$

```
1:  /*To select the offloadable component*/
2:  m=1
3:  while F_CPU(COMP_n) < G_AVG_CPU(CMN):
4:     if F_CPU(COMP_n) < H_CPU(CMN, t_i):
5:        OFFLD(COMP_n) ← 0
6:     else if F_CPU(COMP_n) > H_CPU(CMN, t_i):
7:        OFFLD(COMPn) ← 1
8:     end if
9:  end while
10: return OFFLD_CMN(COMP_n)
```

During the component offloading, the component selection is also important to the efficient component offloading execution. In our approach, the framework selects offloadable component during the runtime dynamically. We present the dynamic component selection algorithm in the Table 3. In the algorithm, the framework compares the CPU utilization value of a component with the predicted CPU utilization value. If the $F_{CPU}(COMP_n)$ is smaller than the predicted $H_{CPU}(CMN, t_i)$ the component will run on the current client mobile node. In contrast, the framework determines to migrate the component to other offloading service node.

Copyright © 2017 SERSC Australia

4. Design Model of the Framework

4.1. Framework Architecture

In this section, we present an overall architecture and key components of the predictive offloading framework. The framework consists of 3 nodes; *Client mobile Tier*, *Offloading Service Node Tier*, and *Dispatcher and Learner Node Tier*. As shown in Figure 1, the predictive offloading framework consists of a number of functional components. A dispatcher and learner node is a stationary computer system with a fixed IP with relatively high computing power and resources. The only responsibility of the node is to maintain the up-to-date status information of nearby offloading service nodes in a directory. The directory, which is used in client mobile nodes is designed to lookup available offloading service nodes and find a most appropriate node as server.

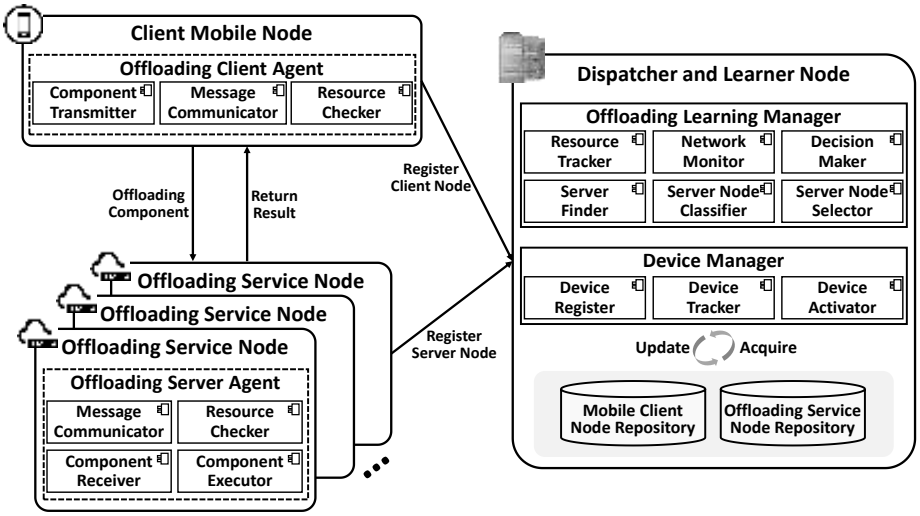

Figure 1. Key Components of the Predictive Offloading Framework

The resource checker component tracks resource status of a client node, just like CPU utilization, network bandwidth, and *etc*. For effective CPU utilization monitoring, the component does not check CPU status continuously. The resource checker component starts to collect CPU status data when an insufficient resource condition event has been captured. The component transmitter migrates some offloadable components to the offloading service node, and the component receiver receives the transmitted offloadable components for further execution. In order to perform message transactions, the message communicator component is responsible to manage those offloading-related data. Message transaction includes input data to execute offloading and return value from offloading service node. On the offloading service node, the component executor executes to receive offloadable components and notify the message communicator to send execution result to client node.

The server node classifier component performs relevant offloading service node clustering function for efficient offloading. The component manages most relevant offloading service nodes to the same cluster for further service node selection. In addition, the server node selector component performs the task to select one or more dedicated offloading service nodes for component migration. The decision maker component analyzes receiving real-time CPU status from the client node, and then predicts the future CPU status trend. The network monitor component collects network bandwidth status of a client node in order to make component offloading decision by interacting with the message communicator. The server finder is responsible for monitoring whether those registered offloading service nodes are available by interacting with the device tracker

 Copyright © 2017 SERSC Australia

component in order to prepare for component offloading.

The device register component is a component that allows resource available mobile devices and desktop PCs to register as offloading service nodes. The framework selects one or more offloading service nodes from registered nodes for component migration. The device tracker manages registered offloading service nodes for ensuring their availability for server node classification and component migration. The device tracker sends a small size data message to the offloading service node for checking their online availability. The device activator component is responsible for activating the selected offloading service node by sending a message to prepare component migration.

4.2. Dynamic Model of Component Offloading

The proposed offloading framework includes three different types of participants and they are responsible for processing different working tasks. To specify overall workflows of the participants, in the Figure 2, an activity diagram is presented which includes client mobile node, dispatcher and learner node, and offloading service node with their interactions.

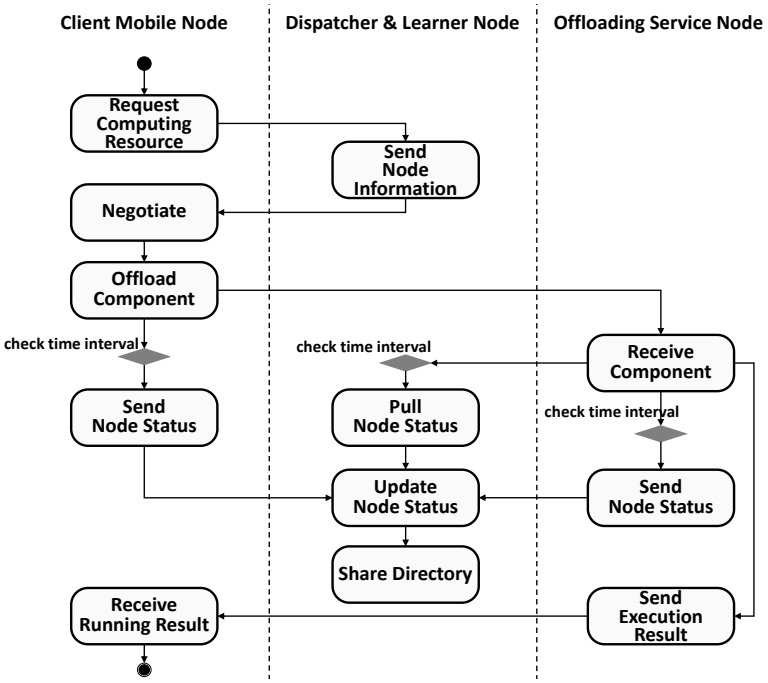

Figure 2. Activity Diagram of Overall Offloading Process

At the beginning of the component offloading, a client mobile node sends a resource request message to its dispatcher and learning node. The request message specifies required computation resources, such as CPU, memory, and network bandwidth. With the retrieved request, the dispatcher and learner node returns some resource available offloading service nodes information to the client mobile node and then the client node starts to negotiate with each offloading service node. During the negotiation, the client node makes decisions on which offloadable components should be migrated and when to migrate. After negotiation work completed, the offloading service node would execute the received offloadable components. During the component execution, both of the client and service nodes send their node status to the dispatcher and learner node respectively. Moreover, the dispatcher and learner node will update node repository and share the node status information with other working dispatcher and learner nodes. We also present a sequence diagram that depicts the method invocations in the offloading processes in Figure 3.

Copyright © 2017 SERSC Australia

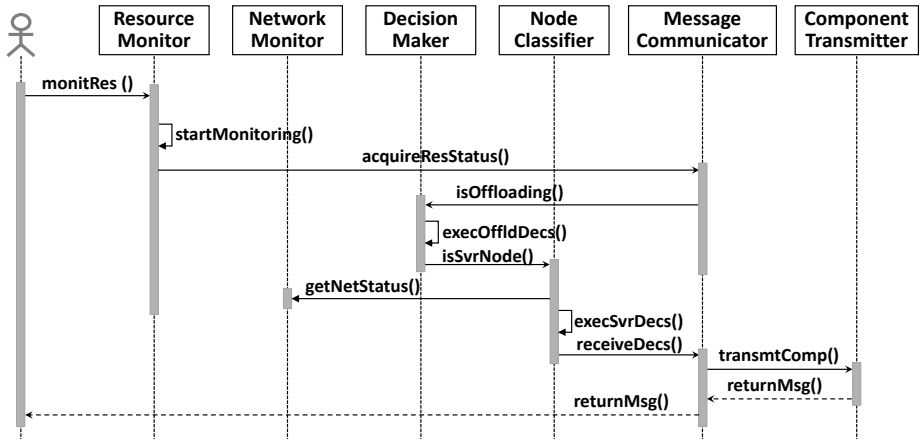

Figure 3. Sequence Diagram of Component Offloading

The proposed framework checks real-time resource status of a client mobile node by calling a *monitRes()* method of the *ResourceMonitor* object to acquire CPU utilization value. The *DecisionMaker* is the object which could make decision on the offloading timing. When the component offloading timing is determined, the *NodeClassifier* object will allocate one or more offloading service nodes to execute offloadable components. The *ComponentTransmitter* object is responsible to migrate selected offloadable component from the client mobile node to those offloading service nodes. At last, the *MessageCommunicator* object sends the execution result of migrated component to the client side.

5. Experiment and Evaluation

5.1. Experiment Scenario

In order to verify the efficiency of the proposed prediction algorithm, we conducted experiments with different mobile devices. We conducted a number of experiments by using an implemented Android program under different circumstances like Figure 4. In this experiment, three different client mobile nodes Huawei P9, Xiaomi Mi 4i, and Galaxy Note5 are selected and the two different dispatcher and learner nodes a dell desktop and a virtual machine are selected to perform the component offloading experiment.

 Copyright © 2017 SERSC Australia

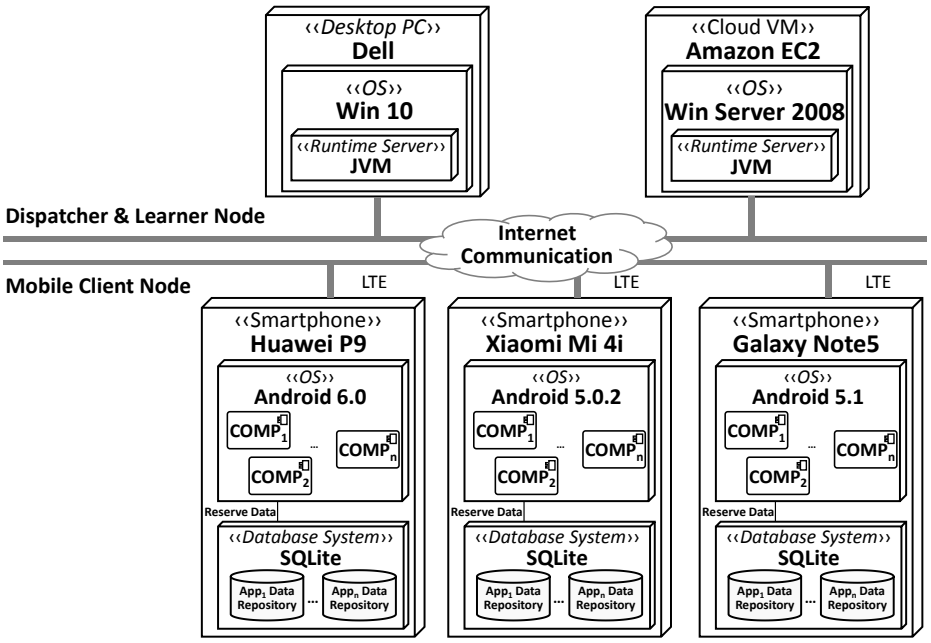

Figure 4. Client Mobile Node and Dispatcher & Learner Node in Experiment

In the experiment, we used a matrix-matrix multiplication program, which is a widely used test program in reducing energy consumption and improving energy efficiency in high performance computing [27]. Due to its characteristics of being easy to increase computational complexity and flexible to change the transmission data size at runtime, we conducted a number of experiments to verify our proposal. Each client node was running totally 12 test cases, as shown in Table 4.

Table 4. Experiment Test Case

	Matrix 1	Matrix 2	Data Size	Complexity
Case 1	100×100	100×100	Small(S)	Low(L)
Case 2	110×100	100×110	Small	Low
		…		
Case 5	300×100	100×300	Medium(M)	Medium(M)
Case 6	310×110	110×310	Medium	Medium
		…		
Case 11	540×130	130×540	Large(L)	High(H)
Case 12	550×140	140×550	Large	High

5.2. Experiment Result

The objective of conducting the experiment is to validate energy efficiency and performance improvement of the proposed framework. The following experiment result is the total time consumption comparisons of three different offloading methods which are non-offloading, traditional dynamic offloading and the proposed predictive offloading as shown in Figure 5.

Copyright © 2017 SERSC Australia

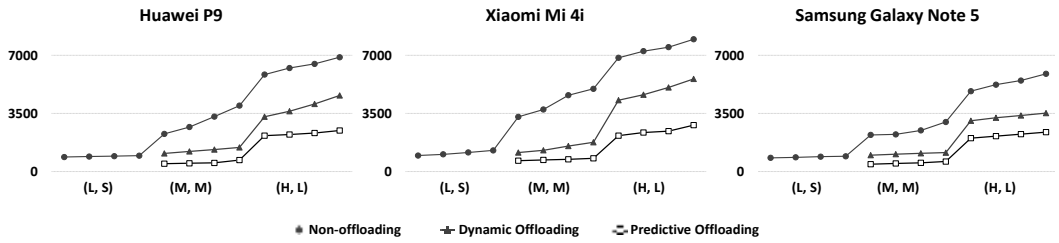

Figure 5. Time Cost Comparison of Non-offloading, Traditional Dynamic Offloading, and Predictive Offloading

With the increasing of matrix dimensions of matrix-matrix multiplication program, both of computation complexity and transmission data size are increased. Among the three situations, offloading begins when computation complexity is in medium. However, due to differential in equipped hardware capability, offloading started time point is slightly different. As computational complexity increasing, the time cost difference is easy to capture by comparing time cost trends among them.

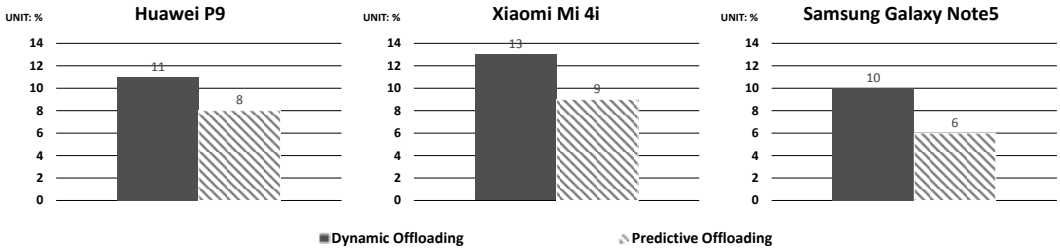

Figure 6. Comparison Result of Battery Usage Efficiency

We also present the battery utilization comparison between dynamic offloading and predictive offloading. As shown in Figure 6, the experiment result presents the total battery consumption degree via executing all of the 12 test cases. With the continuous increasing of computational complexity, the maximum battery consumption value increased up to about 13%. In contrast, in predictive offloading, the maximum battery consumption is lower than 10% which is smaller than the dynamic offloading.

6. Conclusion

Mobile computing has become a common computing paradigm which could provide convenience to people's daily life by offering a number of useful applications. However, inherent defects of mobile device affect the quality of mobile applications. Due to the capability limit of network communication, a remedy scheme, component offloading makes room for the mentioned issues via migrating computing intensive component to cloud server. However, a significant technical challenge is to assure the quality of applications in component offloading. In this paper, the proposed predictive offloading framework with its overall architecture which can provides the chance to perform efficient offloading. Our strategy for the efficient component offloading is to utilize a look-ahead method to identify the need for component offloading. We present an efficient algorithm to predict the demand of resources in advance. At last, we conducted experiments via using matrix-matrix multiplication program to validate the performance improvement.

Acknowledgement

This research was supported by Korea Research Fellowship program funded by the Ministry of Science, ICT and Future Planning through the National Research Foundation of Korea (NRF-2015R1A2A2A01004078).

 Copyright © 2017 SERSC Australia

This paper is a revised and expanded version of a paper entitled *'A Predictive Offloading Framework for Efficient Mobile Cloud Computing'* presented at the 6th International Conference on Next Generation Computer and Information Technology (NGCIT 2017) which was held in Ho Chi Minh City, Vietnam last August 16-18, 2017..

References

[1] S. A. Sanaei, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges", IEEE Communications Survey and Tutorials, vol. 16, no. 1, **(2013)**, pp. 369-392.
[2] D. Huang, T. Xing and H. Wu, "Mobile Cloud Computing Service Models: A User-Centric Approach", IEEE Network, vol. 27, no. 5, **(2013)**, pp. 6-11.
[3] G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing", IEEE Computer, vol. 27, no. 4, **(1994)**, pp. 38-47.
[4] L. Gkatzikis and I. Koutsopoulos, "Migrate or Not? Exploiting Dynamic Task Migration in Mobile Cloud Computing Systems", IEEE Wireless Communications, vol. 20, no. 3, **(2013)**, pp. 24-32.
[5] M. Sharifi, S. Kafaie and O. Kashefi, "A Survey and Taxonomy of Cyber Foraging of Mobile Devices", IEEE Communications Survey and Tutorials, vol. 14, no. 4, **(2011)**, pp. 1232-1243.
[6] A.-N. Moldovan, S. Weibelzahl and C. H. Muntean, "Energy-aware Mobile Learning: Opportunities and Challenges", IEEE Communications Survey and Tutorials, vol. 16, no. 1, **(2013)**, pp. 234-265.
[7] J. Shuja and A. Gani, "A Survey of Mobile Device Virtualization: Taxonomy and State of the Art", ACM Computing Surveys, vol. 49, no. 1, **(2016)**, pp. 1:1-1:36.
[8] S. Abolfazli and Z. Sanaei, "Cloud-based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges", IEEE Communications Survey and Tutorials, vol. 16, no. 1, **(2013)**, pp. 337-368.
[9] N. Fernando, S. W. Loke and W. Rahayu, "Mobile Cloud Computing: A Survey", Future Generation Computer Systems, vol. 29, no. 1, **(2013)**, pp. 84-106.
[10] H. J. La and S. D. Kim, "A Self-Stabilizing Process for Mobile Cloud Computing", 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering (SOSE 2013), **(2013)**, pp. 454-462.
[11] D. Huang, P. Wang and D. Niyato, "A Dynamic Offloading Algorithm for Mobile Comoputing", IEEE Transactions on Wireless Communications, vol. 11, no. 6, **(2012)**, pp. 1991-1995.
[12] X. Ma and Y. Zhao, "When Mobile Terminals Meet the Cloud: Computation Offloading as the Bridge", IEEE Networks, vol. 27, no. 5, **(2013)**, pp. 28-33.
[13] H. Flores and P. Hui, "Mobile Code Offloading: From Concept to Practice and Beyond", IEEE Communications Magazine, vol. 53, no. 3, **(2015)**, pp. 80-88.
[14] H. Wu, D. Huang and M. Chen, "POEM: On Establishing A Personal On-demand Execution Environment for Mobile Cloud Applications", In Proceedings of 2015 IEEE International Conference on Mobile Services (MS 2015), **(2015)**, pp. 41-48.
[15] E. Cuervo and A. Balasubramanian, "MAUI: Making Smartphones Last Longer with Code Offload", In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), **(2010)**, pp. 49-62.
[16] B. G. Chun and S. H. Ihm, "CloneCloud: Elastic Execution between Mobile Device and Cloud", In Proceedings of the Sixth Conference on Computer Systems (EuroSys 2011), **(2011)**, pp. 301-314.
[17] H. J. La and S. D. Kim, "A Taxonomy of Offloading in Mobile Cloud Computing", In Proceedings of 2014 IEEE 7 International Conference on Service-Oriented Computing and Applications (SOCA 2014), **(2014)**, pp. 147-153.
[18] W. Liu and J.-J. Chen, "Computation Offloading by Using Timing Unreliable Components in Real-time Systems," In Proceedings of 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC 2014), **(2014)**, pp. 1-6.
[19] S. Deng and L. Huang, "Computation Offloading for Service Workflow in Mobile Cloud Computing", IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 12, **(2014)**, pp. 3317-3329.
[20] B. Li and Z. Liu, "Mobility Prediction based Opportunistic Computational Offloading for Mobile Device Cloud", In Proceedings of 2014 IEEE 17th International Conference on Computational Science and Engineering (CSE 2014), **(2014)**, pp. 786-792.
[21] T.-Y. Lin and T.-A. Lin, "Context-Aware Decision Engine for Mobile Cloud Offloading", In Proceedings of 2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW 2013), **(2013)**, pp. 111-116.
[22] M. Akram and A. ElNahas, "Energy-Aware Offloading Technique for Mobile Cloud Computing", In Proceedings of 2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud 2015), **(2015)**, pp. 349-356.
[23] S. Yang and D. Kwon, "Techniques to Minimize State Transfer Costs for Dynamic Execution Offloading in Mobile Cloud Computing", IEEE Transactions on Mobile Computing, vol. 13, no. 11, (2014), pp. 2648-2660.
[24] W.-L. Wang, T. L. Hemminger and M.-H. Tang, "A Moving Average Non-Homogeneous Poisson Process Reliability Growth Model to Account for Software with Repair and System Structures", IEEE Transactions on Reliability, vol. 56, no. 3, **(2007)**, pp. 411-421.

Copyright © 2017 SERSC Australia

[25] S. Hansun, "A New Approach of Moving Average Method in Time Series Analysis", In Proceedings of 2013 Conference on New Media Studies (CoNMedia 2013), **(2013)**, pp. 1-4.
[26] A. Raudys, "Optimal Negative Weight Moving Average for Stock Price Series Smoothing", In Proceedings of 2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr 2014), **(2014)**, pp. 239-246.
[27] L. Chen, P. Wu, "Energy Efficient Parallel Matrix-Matrix Multiplication for DVFS-Enabled Clusters", In Proceedings of 2012 21 International Conference on Parallel Processing Workshops (ICPPW 2012), **(2012)**, pp. 239-245.

Authors

Zhen Zhe Piao, he is a Ph.D. candidate in the Department of Computer Science at Soongsil University, Seoul, Korea. He received his bachelor's degree from Changchun Normal University in 2011 and master degree from Soongsil University in 2014. His research interests include cloud computing and mobile cloud computing

Soo Dong Kim, he is a Professor in the department of Computer Science at Soongsil University, Seoul, Korea. He received his B.S. degree in Computer Science from Northeast Missouri State University in 1984, and his Master and Ph.D. degrees from the University of Iowa, Iowa, USA in 1988 and 1991 respectively. His research interests include software reuse, software architecture, cloud computing, and smart services. He has been providing consulting and invited lectures to IT industry and government organizations.

 Copyright © 2017 SERSC Australia

Copyright © 2017 SERSC Australia