JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 9, No. 3, September 2015, pp. 155-162

User Mobility Model Based Computation Offloading Decision for Mobile Cloud

Kilho Lee and Insik Shin*

School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, Korea

khlee@cps.kaist.ac.kr, insik.shin@cs.kaist.ac.kr

Abstract

The last decade has seen a rapid growth in the use of mobile devices all over the world. With an increasing use of mobile devices, mobile applications are becoming more diverse and complex, demanding more computational resources. However, mobile devices are typically resource-limited (i.e., a slower-speed CPU, a smaller memory) due to a variety of reasons. Mobile users will be capable of running applications with heavy computation if they can offload some of their computations to other places, such as a desktop or server machines. However, mobile users are typically subject to dynamically changing network environments, particularly, due to user mobility. This makes it hard to choose good offloading decisions in mobile environments. In general, users' mobility can provide some hints for upcoming changes to network environments. Motivated by this, we propose a mobility model of each individual user taking advantage of the regularity of his/her mobility pattern, and develop an offloading decision-making technique based on the mobility model. We evaluate our technique through trace-based simulation with real log data traces from 14 Android users. Our evaluation results show that the proposed technique can help boost the performance of mobile devices in terms of response time and energy consumption, when users are highly mobile.

Category: Embedded computing

Keywords: Mobile cloud; Computation offloading; Mobility; Cyber physical systems

I. INTRODUCTION

Cyber-physical systems (CPS) are next-generation embedded systems featuring a tight integration of computing and physical elements. Emerging applications of CPS include avionics, automobiles, medical devices, robotics, and consumer electronics. Many of them are mobile in nature. For instance, passengers ride cars for the convenience of moving, patients carry implanted medical devices for health, and people bear mobile phones for a variety of purposes. As such, user mobility is one of the key components of mobile systems that actually move the systems. Thereby, it often entails a good understanding of user mobility in addressing many problems of mobile systems.

Many mobile devices are typically subject to limited resources, such as low computing power, unstable wireless communication systems, and scarce energy capacities. In spite of such limitations, mobile CPS applications are becoming more diverse and complex, requiring heavy computation and network communication. As an example, the Google Glass project [1] proposes next-generation Augment-Reality (AR) based services that combine virtual information with real world images.

Imagine the following scenario. Alice is on the trip to

Open Access http://dx.doi.org/10.5626/JCSE.2015.9.3.155 http://jcse.kiise.org

Received 03 September 2015; **Accepted** 08 September 2015

*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

Europe, and wants to find a nearby fabulous restaurant. She decides to run an AR-based recommendation, then she looks around with wearing the Google Glass. The glasses will show a combination of recommendation information and real restaurant images around her. Since the application involves heavy computation, she finds the application continuously runs on her Google Glass, longer than several tens of seconds, which is unacceptable. So she decides to utilize a new mobile cloud service, the ability for computation offloading. With this new service on, she runs the application again, and OS automatically determines whether or not to offload the major computation of the application (see Fig. 1). If she is in a situation favorable towards computation offloading (i.e., at a hotel with WiFi connection), OS chooses to do it and she can benefit from offloading to get the recommendation quickly. On the other hand, if she were in an unfavorable situation for offloading (i.e., in a bus or in driving), OS would not choose it because offloading could make it even longer to complete the recommendation service compared to no offloading.

The above is a fictional scenario, however, representative of the proposed functionality of mobile clouds, which is dynamic computation offloading. Such dynamic computation offloading essentially raises many technical challenges. It especially entails a mobile computation platform which provides an offloading mechanism and offloading decision-making policy for dynamic computation offloading. Existing studies [2-4] are focusing on offloading mechanisms, providing a technical basis for computation offloading, such as programming models, run-time environments, and program structure analysis techniques. Beyond those offloading mechanisms, offloading a decision-making policy is important to provide beneficial computation offloading.

Mobile network environments have a great influence on the performance of computation offloading. For example, if a mobile device has a stable network connectivity and plenty of network bandwidth, then computation offloading will result in better performance in terms of both response time and energy consumption. However, mobile users are typically subject to dynamically changing environments due to their mobility. Thus, a high-quality decision requires a good understanding of network condition changes and taking near-future network conditions into account to make a decision, whereas user mobility makes it hard to predict. Thereby, this paper aims to have a good understanding of user mobility and to incorporate it into good offloading decision-making. Then it proposes an offloading decision-making technique based on users' mobility model.

II. BACKGROUND

A. Related Work

Mobile cloud. Recently, a few studies [2-4] have been reported for development of computation offloading frameworks in mobile environments, focusing on offloading mechanisms. For example, MAUI [2] proposes a dynamic offloading framework with its own run-time mechanism. It first requires explicit user annotation specifying which methods can be offloaded. For instance, methods should not be marked for offloading if they make use of native function calls, such as device-specific function calls. It then profiles the execution time of offload-able methods, both when they run on a mobile device and on cloud, respectively. MAUI makes offloading decision offline through integer linear programming (ILP) optimization based on the profiled execution time and measured network quality. Similar to MAUI, CloneCloud [3] proposes its own computation offloading framework. The main difference between them is that CloneCloud considers migrating an entire virtual machine, while MAUI considers offloading a unit of computation (i.e., function/method). CloneCloud thereby proposes a modified Dalvik VM [5] as a run-time, and it does not need explicit annotation for distinguishing offload-able computation. Odessa [4] aims to support applications which

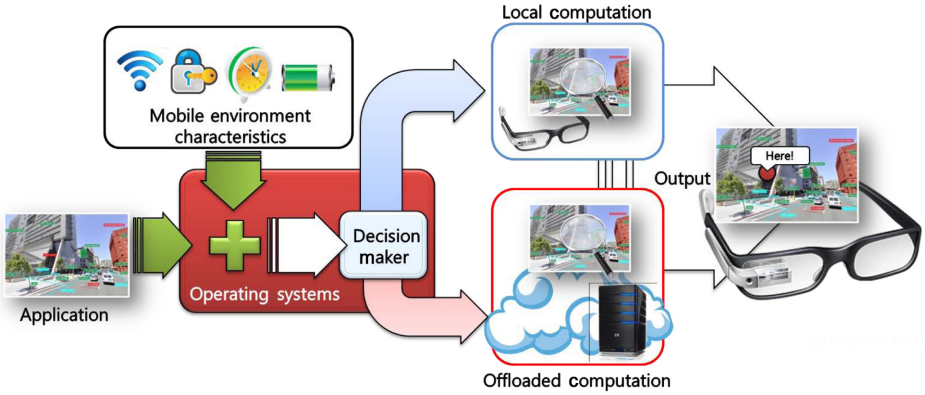

Fig. 1. Overview of dynamic computation offloading.

have dependencies between data flows, and it supports to offload a portion of parallel executions to maximize parallelism.

User mobility. Mobility modeling technique has been widely used to predict changing environments of mobile systems. In order to measure and to improve the performance, wireless ad-hoc network systems use the mobility model. A few works [6-8] focus on building a mobility model based on the distribution of users (nodes), and predicting future distribution of the users. Another work [9] focuses on building a mobility model of a certain user, and predicting the user's location. The previous works cover how to predict the location of the users.

B. Computation Offloading

In our system, both a mobile machine and a remote machine (i.e., a server in the cloud) have the same program logic, but the modules on the remote are faster. When the mobile machine wants to offload a computation, it transfers input data to a remote machine, triggering the execution of a corresponding module on the remote machine, and then receives the resulting data back. We define such a sequence of operations as an *offloaded* computation. In contrast, when the mobile machine executes a module on the machine itself, we define it as a *local* computation. Under the above definition, the response time of an *offloaded* computation is defined as the sum of input/result data transmission time and the execution time of a method running on a remote machine. This paper does not focus on how to estimate an exact execution time of each method on local or remote machines. We assume that the decision maker can apply the state of the art [2, 3] profiling techniques to get those parameters.

C. Problem Statement

The main goal of this work is to develop a good policy for offloading decision-making. An offloading decision is considered as good if an *offloaded* computation runs faster than a *local* computation. In order to meet the goal, the technique should address the following challenges, making high-quality decisions under dynamically changing mobile network conditions, in an energy efficient way.

III. MOBILITY-AWARE OFFLOADING DECISION-MAKING

We propose a mobility-aware offloading decision maker, named *Mob-aware*, which takes into account near-future network changes based on the user's mobility. The *Mob-aware* decision maker gathers previous user movements and network changes corresponding to the movements, builds mobility model with gathered data, and then makes offloading decisions.

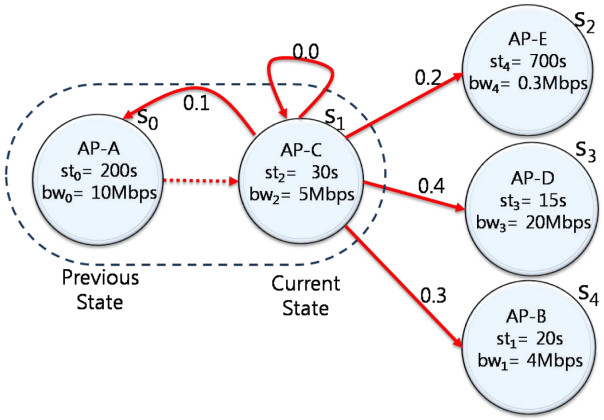

Fig. 2. Mobility model. st_i and bw_i indicate staying time and bandwidth on a state s_i, respectively. An edge weight represents a probability of moving to a certain access point (AP) from the current AP.

A. Mobility Modeling

We propose a mobility model, which reflects a certain user's mobility patterns. By using the mobility model, *Mob-aware* decision maker predicts near-future network condition changes. User mobility often has some regularity [8, 9], and this can provide some hints for what kind of changes can occur to the network in the near future. This motivates modeling of user's mobility patterns.

In our technique, user's mobility is characterized by a sequence of networks to which users are connected. For example, if a user is connected to WiFi, the location of the user is specified by its WiFi access point (AP) ID, then the trajectory of a user is represented as a sequence of WiFi access point IDs. Based on such data, a 2nd-order Markov model was built as a mobility model (see Fig. 2) and trained with mobility patterns of a certain user. The model represents the probability of visiting certain APs in the near future subject to the currently associated AP, an expected network quality under each AP, and an expected staying time under each AP. Each user shows distinct mobility patterns, thus every user has an individually trained mobility model. Fig. 2 shows an instance of the mobility model. Each vertex represents each state, modeled by WiFi APs. Each edge represents hand-over between APs, and a weight of an edge means a probability of moving to a certain AP from the current AP. In each state s_i, bw_i and st_i represent average network bandwidth and average staying time under a certain AP, respectively.

B. Prediction Engine

With the mobility model trained individually, the *Mob-aware* decision maker can predict how a user moves from

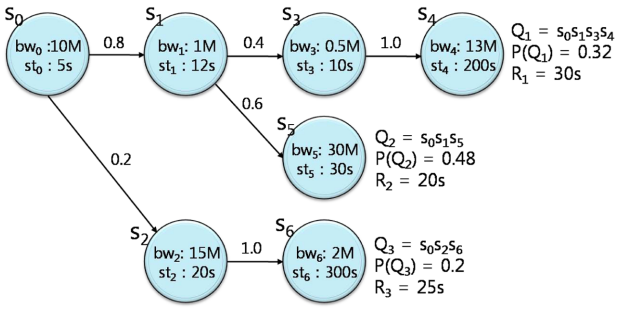

Fig. 3. Prediction engine. Q_i indicates each possible path, $P(Q_i)$ represents the probability of taking a path Q_i. It calculates expected response time of an *offloaded* computation considering every possible path.

the current location. We propose a prediction engine, which predicts near-future network condition based on the mobility model, and calculates an expected response time of a computation. The prediction engine estimates expected response time of an *offloaded* computation and a local computation, respectively. An *offloaded* computation especially involves data transmission, where the engine has to estimate the data transmission time based on predictions of near-future network conditions. The prediction engine explores every possible path in the mobility model. For each possible path, the engine calculates a response time of the computation based on expected network throughput, bw_i, and expected staying time on each AP, st_i, in the model. Then the engine calculates probabilities of the user taking each possible path. Based on these calculations, it estimates expected response time considering every possible path. Fig. 3 depicts the prediction engine. Let Q_i be a possible path (a plausible sequence of APs), and s_k be a state of model. The probability of taking a path Q_i (denoted by $P(Q_i)$) can be derived by Eq. (1) based on the Markov assumption [10].

$$P(Q_i) = P(s_0,\ s_1,\ s_2,\ \ldots,\ s_n) = \prod_{k=2}^{n} P(s_k | s_{k-1},\ s_{k-2}) \quad (1)$$

$$R = \sum_i P(Q_i) \times R_i(Q_i,\ d_u,\ d_r) \quad (2)$$

The prediction engine estimates expected response time of the *offloaded* computation, R, through Eq. (2); where $P(Q_i)$ is a possibility of that the user takes Q_i as his/her future trajectory, and R_i is the response time of the *offloaded* computation on the trajectory Q_i. Since each Q_i consists of a sequence of states s_k, the engine can calculate response time R_i of an *offloaded* computation on the path Q_i by using st_k, bw_k, d_u, and d_r. The prediction engine starts with a sequence only containing the current AP state. It checks whether *offloaded* computation can be finished by $\Sigma_k\, st_k$ in a sequence of states. If not, the engine extends the sequence by adding a next reachable AP state. In the mobility model, typically an AP state has multiple next states. Therefore, the engine makes multiple extended sequences. The engine repeatedly conducts completion checking and sequence extending, and finds every possible Q_i. It then calculates expected response times of the *offloaded* computation, R, with possible paths Q_i and response times on the path R_i.

Path Pruning. If the *offloaded* computation cannot be finished on a certain state sequence, the engine extends the sequence with the next reachable APs in the model. As a consequence, the number of state sequences increases exponentially. It incurs severe computation overheads. The engine restricts the length of each possible state sequence, to avoid the state explosion problem. Since $P(Q_i)$ is defined as multiplication of state transition probability (<1.0), where the longer Q_i has the smaller weight $P(Q_i)$. If the length of a certain state sequence reaches a pruning threshold, the engine changes staying time of the last state in the state sequence to infinite, so that the length of each state sequence is restricted.

C. Adaptive Decision-Making

After calculating R, the prediction engine compares R with the response time of the local computation. Finally, it chooses the faster one as an offloading decision. After the prediction engine makes an offloading decision, either an *offloaded* or a *local* computation runs. While a computation is running, an unexpected move of a user or rapid network condition changes can occur. For example, when a WiFi connection is suddenly disconnected, an *offloaded* computation will suffer severe performance degradation. In order to alleviate such a problem, our decision maker adaptively responds to dynamic network changes, periodically identifying any changes to network quality, an associated AP, and the portion of completed computation.

IV. EVALUATION

We collect usage logs of smartphones from 14 users. We then implement a trace-based simulator to evaluate proposed mobility-aware offloading decision-making techniques. This section presents experiment environments, evaluation tools, and evaluation results.

A. Experiment Environments

Baselines. We propose *Mob-aware* decision maker, which takes network changes into account with mobility model. In order to show an effect on the performance of the *Mob-aware* decision maker, we consider other decision makers, *Net-aware* and *Dual* decision makers, as baselines. *Net-aware* decision maker takes advantage of the current network conditions, under the assumption that the current network conditions would not change much in

the near future. It estimates the data transmission time of an *offloaded* computation by using current network throughput. *Dual* decision maker allows a computation to run on local and remote simultaneously. It takes the *offloaded* computation only when it finishes faster than the *local* computation. Thus, it always makes an optimal decision in terms of response time. In this evaluation section, we present comparison between those different decision-making policies according to the response time and energy consumption.

Trace collecting. In order to gather real user mobility data, we implement an Android logger application that collects log data traces. The traces of each user are used for building an individual mobility model, and those are used for evaluating the performance of decision-making policies. The logger periodically collects log data, which includes a time stamp, network connectivity, GPS coordinates, WiFi status and network throughput. The logger was deployed for 14 users (6 undergraduate students and 8 graduate students), and data traces were collected for at least 3 weeks per user. They consist of 3,770 hours of traces, as well as about 4 million log records.

Trace-driven simulation. Our technique is evaluated through trace-driven simulation. We gathered user data traces including user locations and network conditions. The simulator then builds a mobility model of each user for *Mob-aware* decision maker with gathered data. Upon the traces and the models, the simulator repeatedly carries out the following steps:

1) the simulator chooses a time instant in data traces randomly;

2) at the chosen time instant, each decision maker makes an offloading decision for a given computation;

3) according to each decision at step 2, the simulator calculates the response times of the computation.

B. Simulator Implementation

We implement a simulator, which consists of a decision maker and a response time calculator. The decision maker in the simulator reads data traces and builds a mobility model for each user. After that, it can make a decision based on the mobility model. The response time calculator also reads data traces, especially network throughput changes over time. It then calculates the response time when a given computation follows the decision made by the decision maker.

For a certain moment on data traces, the simulator makes an offloading decision for a computation and calculates the response time of the computation. Thus it can evaluate the performance of the decision-making technique. Clearly, the offloading decision is considered good when the decision maker makes an *offload* decision and the response time of the *offloaded* computation is shorter than the response time of the *local* computation.

Table 1. Power model

	CPU : Idle (mW)	CPU : Active (mW)
WiFi	1788	2415
3G	1377	2473
No network	554	1629

The simulator can calculate response times, and can also calculate energy consumption based on a power model. Table 1 describes the power model according to machine states. The state of a machine depends on usage of CPU, WiFi, and 3G, and is measured by a Monsoon power monitor [11] with actual power consumption of Samsung Nexus S [12].

C. Simulation Results

A case study. *Mob-aware* decision maker comes up with better decisions than others, especially, when a user is moving around and the network condition is thereby changing rapidly. For example, when WiFi hand-over occurs, the network throughput rapidly decreases. Fig. 4 is a piece of collected traces, which depicts such a hand-over case. It shows network throughput changes over time. At 5 seconds, the mobile network suddenly loses its connectivity due to WiFi hand-over.

We perform a simulation on this case with a dummy computation, which has 30 Mbits and 10 Mbits of input and output data sizes, 1 second of remote execution time, and 10 seconds of local execution time.

At t_0, each decision maker calculates the expected response time of an *offloaded* computation, and decides whether or not to offload the computation. At t_0, *Mob-aware* and *Net-aware* predict 23.4 seconds and 7.8 seconds as the response times of the *offloaded* computation, respectively. Thus, *Mob-aware* decides not to offload, but

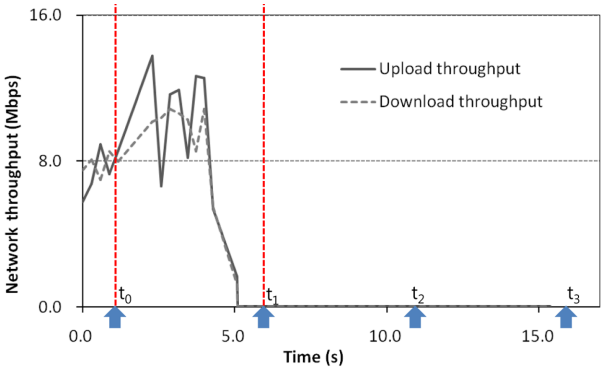

Fig. 4. Decision making with WiFi handover. It shows throughput changes and timing instants corresponding to decision makings. Upon this given trace, *Mob-aware* finishes given computation at t_2, while *Net-aware* finishes the computation at t_3.

Net-aware decides to offload the computation. Even though the current network condition is good enough to offload, *Mob-aware* considers that it is possible to lose network connectivity based on a mobility model of the user. At t_1, both decision makers conduct adaptive decision-making. Since the network condition is getting worse, *Mob-aware* maintains its *local* decision, but *Net-aware* changes its *offload* decision into *local*. Therefore, *Mob-aware* and *Net-aware* have 10 and 15 seconds of response times, respectively.

As shown in Fig. 4, *Mob-aware* decision maker can reach a better decision, especially when network conditions rapidly change due to user mobility. In order to evaluate the performance of *Mob-aware* policy more extensively, we conduct large scale simulations over millions of real user log records.

Various workloads. Within traces of each user, the simulator chooses time instants for an offloading decision in a way that it selects 3 time instants randomly within each interval of a single WiFi AP association. Finally, 23,637 time instants are selected. In order to evaluate the performance of each decision-making policy under various workload setup, we conduct simulations on selected time instants while changing the computation workload. We use a dummy computation specified by input size, output size, remote execution time and local execution time. We changed the input size from 1 MB to 10 MB. The output size and the remote execution time are changed depending on the input size. The range of the output size is set from 0.33 MB to 3.3 MB (one third of input size), and the range of the remote execution time is set from 0.25 to 2.5 seconds (4 MB/s). The local execution time is fixed to 10 seconds. Therefore, the higher input size brings the longer response time of *offloaded* computation, and makes it difficult to benefit from the *offloaded* computation.

Fig. 5 shows the average response times while the input size changes. The x-axis represents input sizes of the dummy computation. As aforementioned, output sizes and remote execution times are changed depending on input size changes. Each plotted point represents an average response time of each decision-making policy on 23 K selected time instants. In the result, *Mob-aware* results better average response times than *Net-aware*. Since *Mob-aware* makes decisions based on mobility predictions, it can produce more proper offloading decision than *Net-aware*.

Fig. 6 shows average energy consumptions while the input size changes. The x-axis represents input sizes, and each plotted point represents an average energy consumption of each decision-making policy. In the result, *Mob-aware* is slightly better than *Net-aware*. *Dual* consumes twice more energy than *Mob-aware* and *Net-aware*, since it runs *offloaded* and *local* computations concurrently. Although *Dual* makes optimal performances in terms of response time, the result confirms that *Dual* is an inefficient policy in terms of energy savings.

Various network conditions. We perform another experiment to show effectiveness of our approach when network condition is highly dynamic. We select 23,637 time instants identical to the previous experiment. We perform the simulations with the computation that has 30 Mbits of input data size, 10 Mbits of result data size, 1 second of remote execution time, and 10 seconds of local execution. We also use adaptive decision-making techniques with 5 seconds periods. At every time instant selected, the simulator makes offloading decisions with three different policies: *Mob-aware*, *Net-aware*, and *Dual*. It then calculates the response time of the computation according to each decision.

In order to characterize the degree of mobility of each experiment case, results are broadly classified into a couple of categories according to the number of WiFi handover occurrences during each computation run. User mobility is considered as *high* if it experiences more handovers. In general, when user mobility becomes higher, the network condition goes less stable and it makes it more difficult to make a good offloading decision.

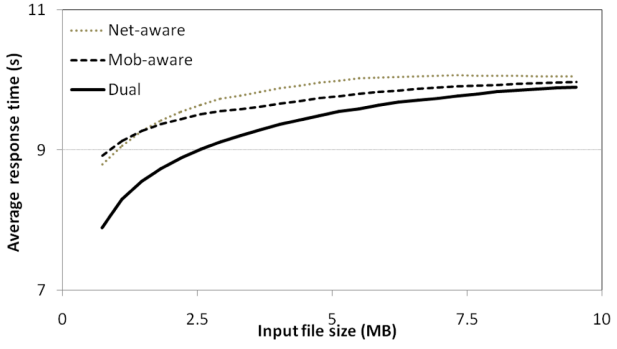

Fig. 5. Average response time on various workloads. *Mob-aware* results better response time than *Net-aware*, and *Dual* results optimal response time.

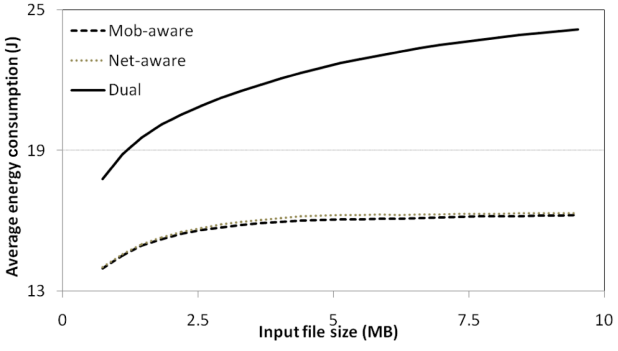

Fig. 6. Average energy consumption on various workloads. Although *Dual* results optimal response time, *Dual* uses energy twice as much as *Mob-aware* and *Net-aware* use.

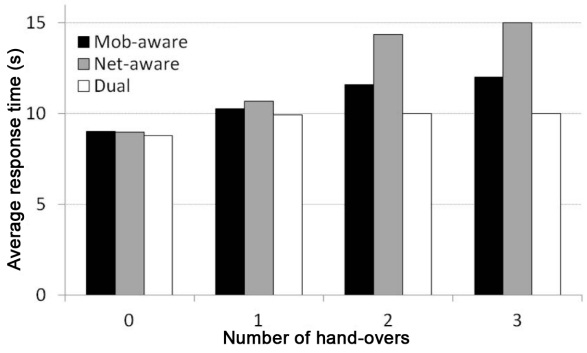

Fig. 7. Average response time. It compares different decision making policies in terms of response time. When user mobility becomes higher (going through more than three WiFi APs), *Mob-aware* shows around 20% better performance than *Net-aware.*

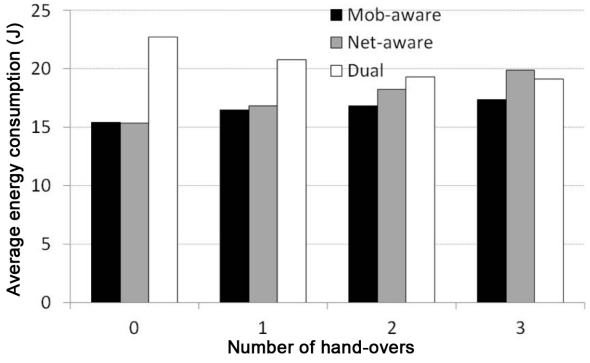

Fig. 8. Average energy consumption. The figure shows that *Mob-aware* is able to save 7%-12% more energy than *Net-aware* when user mobility is high.

Fig. 7 depicts the performance of different decision-making policies in terms of the response time of computation. The x-axis represents the number of hand-over occurrences, and the y-axis represents the average response time. The figure shows that *Mob-aware* and *Net-aware* provide the average response times comparable to each other when user mobility is low. However, *Mob-aware* significantly outperforms *Net-aware* by around 20%, when user mobility is high. It indicates that the network condition becomes unstable when user mobility is high and *Mob-aware* makes better decisions than *Net-aware* in this case. This result implies it is thereby important to consider mobility to make a proper offloading decision in mobile environments.

Fig. 8 plots the average energy consumption of a given computation for different decision-making policies. The x-axis indicates the degree of user mobility in terms of the number of hand-overs, and the y-axis represents the average energy consumption of a given computation. The figure shows that *Dual* consumes more energy than *Mob-aware* and *Net-aware.* This is because *Dual* always employs both *local* and *offloaded* computations simultaneously, while *Mob-aware* and *Net-aware* have either a *local* or an *offloaded* computation at a time. The figure also shows *Mob-aware* consumes 7.7%–12.6% less energy than *Net-aware* when user mobility is high. This is interesting because Mob-aware provides even smaller response times than *Net-aware* in the same cases. This is because *Net-aware* changes its decisions more frequently than *Mob-aware* when network conditions are unstable with high user mobility.

V. CONCLUSION

This paper proposes a mobile computation offloading technique, focusing on user mobility-aware decision-making, which can predict near-future network condition through user mobility models. Our evaluation results show that our technique can be particularly beneficial when users are moving around, causing fluctuations in network quality. In this paper, we consider only the trajectory as users' mobility. However, other factors in users' mobility, such as moving speed, may also affect offloading performance. Furthermore, computation offloading can show different behavior depending on context of user's location. Thereby, we plan to extend our technique by considering such factors, for example, developing a location-aware decision-making framework.

REFERENCES

1. "Google glass project," http://www.google.com/glass/start/.
2. E. Cuervo, A. Balasubramanian, D. K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, 2010, pp. 49-62.
3. B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the 6th Conference on Computer Systems*, Salzburg, Austria, 2011, pp. 301-314.
4. M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, Bethesda, MD, pp. 43-56.
5. D. Bornstein, "Dalvik VM internals," http://sites.google.com/site/io/dalvik-vm-internals.
6. D. Bhattacharjee, A. Rao, C. Shah, M. Shah, and A. Helmy, "Empirical modeling of campus-wide pedestrian mobility observations on the USC campus," in *Proceedings of 2004 IEEE 60th Vehicular Technology Conference (VTC2004-Fall),* Los Angeles, CA, 2004, pp. 2887-2891.
7. T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483-502, 2002.
8. W. Su, S. J. Lee, and M. Gerla, "Mobility prediction in wire-

less networks," in *21st Century Military Communications Conference Proceedings (MILCOM2000)*, Los Angeles, CA, 2000, pp. 491-495.
9. T. Liu, P. Bahl, and I. Chlamtac, "Mobility modeling, location tracking, and trajectory prediction in wireless ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 6, pp. 922-936, 1998.
10. B. Everitt, *The Cambridge Dictionary of Statistics,* 2nd ed. Cambridge, UK: Cambridge University Press Cambridge, 2002.
11. "Monsoon power monitor," http://www.msoon.com.
12. Samsung, "Meet the Nexus S with Android 2.3," http://www.samsung.com/uk/discover/meet-the-nexus-s-with-android-2-3.

Kilho Lee

Kilho Lee is a Ph.D. student in School of Computing at KAIST, South Korea. He received a B.S. in Information and Computer Engineering from Ajou University, South Korea in 2010, and an M.S. in Computer Science from KAIST, South Korea in 2012. His research interests lie in mobile computing and mobile networking systems

Insik Shin

Insik Shin is an associate professor in School of Computing at KAIST, where he joined in 2008. He received a B.S. from Korea University, an M.S. from Stanford University, and a Ph.D. from University of Pennsylvania all in Computer Science in 1994, 1998, and 2006, respectively. He has been a post-doctoral research fellow at Malardalen University, Sweden, and a visiting scholar at University of Illinois, Urbana-Champaign until 2008. His research interests lie in mobile computing, real-time embedded systems, and cyber-physical systems. He has served various program committees in real-time embedded systems, including RTSS, RTAS, ECRTS, and EMSOFT. He received best paper awards, including the Best Paper awards from RTSS in 2003 and 2012, the Best Student Paper Award from RTAS in 2011, the Best Paper award from CPSNA in 2014, and Best Paper runner-ups at RTSS and ECRTS in 2008.