User Mobility-Aware Decision Making for Mobile Computation Offloading

Kilho Lee and Insik Shin
Dept. of Computer Science, KAIST, Republic of Korea
{kh.lee,ishin}@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 getting 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 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 make good offloading decisions in mobile environments. In general, user's 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 result shows that the proposed technique can help mobile devices to boost its performance in terms of response time and energy consumption, when users are highly mobile.**

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 moves 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 powers, instable wireless communications, and scarce energy capacities. In spite of such limitations, mobile CPS applications are getting more diverse and complex, requiring heavy computation and network communication. As an example, Google Glass project [1] proposes next-generation AR (Augment-Reality) based services that combine virtual information with real world images.

Since these applications usually involve heavy vision processing, it may take tens of seconds, which is hardly acceptable. The device can apply dynamic computation offloading to the heavy application. With dynamic computation offloading, if the device is in a situation favorable towards computation offloading (i.e., at a hotel with WiFi connection), OS chooses to do it and the application can benefit from offloading to get the result quickly. On the other hand, 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 computation compared to no offloading.

Such dynamic computation offloading essentially raises many technical challenges. Especially, it entails a mobile computation platform which provides offloading mechanism and offloading decision making policy for dynamic computation offloading. Existing studies [2], [3], [4] are focusing on offloading mechanisms, providing technical basis for computation offloading such as programming models, run-time environments, and program structure analysis techniques. Beyond those offloading mechanisms, offloading 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 condition into account to make a decision, while 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 user's mobility model.

II. Background

A. Related works

Recently, a few studies [2], [3], [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 their own run-time mechanism. It first requires explicit user annotation specifying which methods are allowed to 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 ILP (integer linear programming) optimization based on the profiled execution time and measured

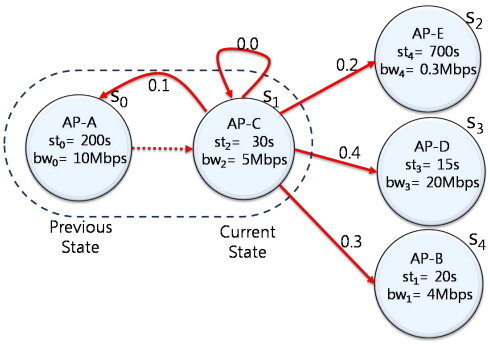

Fig. 1. Mobility model. st_i and bw_i indicate staying time and bandwidth on a state s_i, respectively. An edge weight represents a probability of moving to a certain AP from the current AP.

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

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. The modules on the remote are faster. When the mobile machine wants to offload a computation, it transfers input data to a remote machine, triggers the execution of a corresponding module on the remote machine, and receives result 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 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.

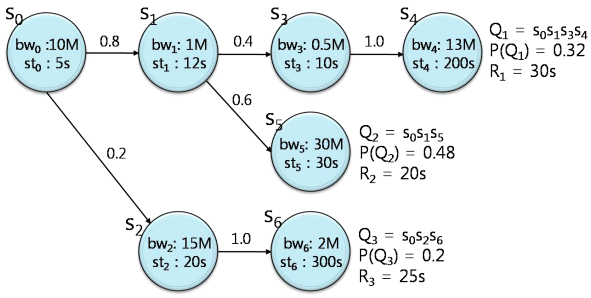

Fig. 2. Prediction engine. Q_i indicates each possible path, $P(Q_i)$ represents the probability of taking a path Q_i. It calculates expected response time of an *offloaded* computation considering every possible path.

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 [6], [7], 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, we build a 2nd-order Markov model as a mobility model (see Figure 1) and train the model 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 a distinct mobility patterns, thus every user has an individually trained mobility model. Figure 1 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 the current location. We propose a prediction engine, which predicts near-future network condition based on the mobility model, calculates expected response time of a computation. The prediction engine estimates expected response time of an *offloaded* computation and a *local* computation, respectively. Especially, an *offloaded* computation involves data transmission, 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 that the user takes each possible path. Based on these calculations, it estimates expected response time considering every possible paths. Figure 2 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 [8].

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

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

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. The engine predicts network throughput changes on the trajectory Q_i by using bw_i and st_i in the model. Then the engine calculates each R_i on Q_i with given input and result data size (denoted by d_u and d_r, respectively).

C. Adaptive decision making

After calculating R, the prediction engine compares the R with the response time of the *local* computation. Finally, it chooses the faster one as an offloading decision. After the prediction engine makes offloading decision, either an *offloaded* or a *local* computation runs. While a computation is running, 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 implement a trace-based simulator to evaluate our proposed mobility-aware offloading decision making technique. This section presents evaluation methodology, evaluation tools, and experimental environments, and discusses results.

A. Experimental methodology

Comparative evaluation. We propose *Mob-aware* decision maker which takes network changes into account with mobility model. In order to show effect on performance of *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-based simulation. Our technique is evaluated through trace-based simulation. We gathered user data traces including user locations and network conditions. The simulator then builds mobility model of each user for *Mob-aware* decision maker by using 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. Data logging

In order to gather real user mobility data, we develop 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 include a time stamp, network connectivity, GPS coordinates, WiFi status and network throughput. The logger was deployed to 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.

C. Trace-based simulator

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 considered as 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.

The simulator not only calculates response times, but also calculates energy consumption based on a power model. Table I describes the power model according to machine states. A state of a machine depends on usage of CPU, WiFi, and 3G. It is measured by Monsoon power monitor [9] with actual power consumption of Samsung Nexus S [10].

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

TABLE I
POWER CONSUMPTION (MW)

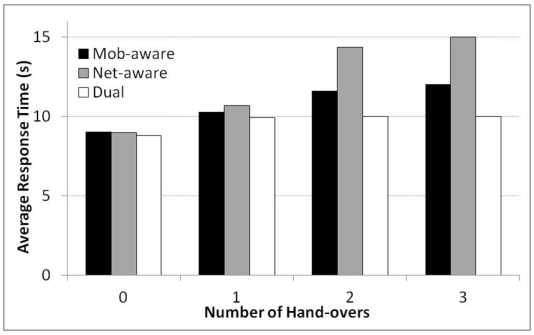

Fig. 3. Average response time. It compares different decision making policies in terms of response time. When user mobility becomes higher (going through more than three WiFi APs), *Mob-aware* shows around 20% better performance than *Net-aware*.

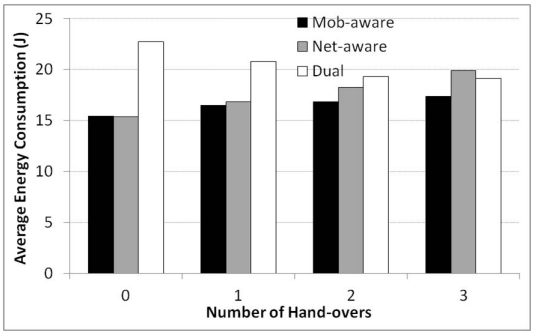

Fig. 4. Average energy consumption. The figure shows that *Mob-aware* is able to save 7%-12% more energy than *Net-aware* when user mobility is high.

D. Simulation results

Large scale simulations. For traces of each user, the simulator chooses time instants for offloading decision making in a way that it selects 3 time instants randomly within each interval of a single WiFi AP association. Finally, 23,637 of time instants are selected. We perform the simulations with the computation that has 30Mbits of input data size, 10Mbits of result data size, 1s of remote execution time, and 10s of local execution. We also use adaptive decision making technique with 5s 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 hand-over occurrences during each computation runs. User mobility is considered as *high* if it experiences more hand-overs. 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.

Figure 3 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.

Figure 4 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 fluctuation in network quality. In this paper, we consider only trajectory as user's mobility. However, other factors in user's 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. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *MobiSys*, 2010.

[3] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *EuroSys*, 2011.

[4] M. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *MobiSys*, 2011.

[5] D. Bornstein, "Dalvik vm internals," http://sites.google.com/site/io/dalvik-vm-internals.

[6] W. Su, S. Lee, and M. Gerla, "Mobility prediction in wireless networks," in *MILCOM*, 2000.

[7] T. Liu, P. Bahl, and I. Chlamtac, "Mobility modeling, location tracking, and trajectory prediction in wireless atm networks," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 6, 1998.

[8] B. Everitt and A. Skrondal, *The Cambridge dictionary of statistics*. Cambridge University Press Cambridge, 2002.

[9] "Monsoon power monitor," http://www.msoon.com.

[10] "Samsung nexus s," http://www.samsung.com/us/article/meet-the-nexus-s-with-android-2-3.