IEEE Mobile Services, Anchorage, Alaska, USA, 29th June 2014

Mirroring Mobile Phone in the Clouds

Bin Ye
School of Computing
The University of Kent
Canterbury, UK
e-mail: by30@kent.ac.uk

Frank Wang
School of Computing
The University of Kent
Canterbury, UK
e-mail: frankwang@ieee.org

***Abstract*— This paper presents a framework of Mirroring Mobile Phone in the Clouds (MMPC) to speed up data/computing intensive applications on a mobile phone by taking full advantage of the super computing power of the clouds. An application on the mobile phone is dynamically partitioned in such a way that the heavy-weighted part is always running on a mirrored server in the clouds while the light-weighted part remains on the mobile phone. A performance improvement (an energy consumption reduction of 70% and a speed-up of 15x) is achieved at the cost of the communication overhead between the mobile phone and the clouds (to transfer the application codes and intermediate results). Our original contributions include a dynamic profiler and a dynamic partitioning algorithm compared with traditional approaches of either statically partitioning a mobile application or modifying a mobile application to support the required partitioning.**

Keywords- Mobile Service, Cloud computing, Virtual Machine, Green Computing

I. INTRODUCTION AND BACKGROUND

In recent years, mobile phones are getting faster and more intelligent, thanks to more powerful processors. However, one of the biggest obstacles for future growth is battery. Unfortunately, technologies trend for batteries indicate that these limitations are here to stay and that energy consumption will remain the primary bottleneck for handheld mobile devices[1]. Making smartphones last longer are becoming mobile industry's foremost challenge [2].

One of most popular technique to reduce energy consumption is dynamic remote execution: application automatically offloads a part of code from mobile phones to Cloud computing servers. In recent years, there are many attempts using two of the following methods. The first approach is to rely on partition algorithm. The partition decision is made based on network condition and whether computing is intensive [3]. The partition scheme is made by a complex algorithm according to each smartphone's environment. The second approach is to clone a virtual mobile on Cloud servers and smartphones do not send any code from mobile to Cloud. Instead, it invokes service located on Cloud servers based on mobile phones conditions [4].

In this paper we present a new framework called MMPC (Mirroring Mobile Phone in the Clouds) that mirrors an entire mobile phone operating system to Cloud servers and uses an intelligent partition algorithm to decide how to deploy an application. MMPC uses DISPY[5] to offload a part of code from mobile phone to Cloud. Firstly, MMPC creates a virtual mobile system on Cloud servers, which brings in a number of advantages. For instance, your personal information can be saved if you lose your phone. It is also possible to have minimal amount of code modifications, leading to an improvement in the development efficiency. Secondly, MMPC uses a profiler to decide how to deploy an application based on the network and mobile environment and then uses DISPY to send a part of application code from the mobile phone to Cloud servers. Thirdly, the virtual mobile system executes the received code. Finally, the cloned system uses DISPY again to send back results to the mobile phone.

II. SYSTEM ARCHITECTURE

A. *System Architecture*

The goal of MMPC is to reduce energy consumption and to improve the performance by using Cloud Computing. This section provides a high-level overview of the architecture, in which a virtual mobile operation system is mirrored and running on a Cloud Computing infrastructure. In this work, DISPY is used to support remote parallel computing. DISPY is a framework developed in python, for parallel execution of computations by distributing them across multiple processors on a single machine, or among many machines in a cluster/grid/cloud. The project set up a 3 nodes cluster. Each node runs a virtual mobile system. Then, DISPY clients are installed on all of the virtual mobile systems. DISPY framework automatically allocates resources to machines.

Figure 1 shows the architecture of MMPC, which is composed of three functional parts. Cloud computing provides an unlimited computing resource. The system is designed to automatically partition a single application into a distribution of execution in such a way that the resource intensive part is run in a powerful clone. The Cloud servers pay the cost of execution including energy and computer resources. The idea is inspired by CloneCloud [4], in which an application is partitioned statically providing a fine-grained partition environment. Unlike CloneCloud, our original contributions include a dynamic profiler and a dynamic partitioning algorithm. In theory, the project

provided a partition strategy to manage code in method level. The designation allows developers to manage their code at the method level in order to decide which part of the code is better to be executed remotely. A fundamental design goal is to allow a fine-grained flexibility on what to run where. The second goal is to reduce the difficulties and complexity of making a partition strategy for a particular application [6]. Another benefit is that if the smart-phone is lost or stolen, the clone image can be used as backup.

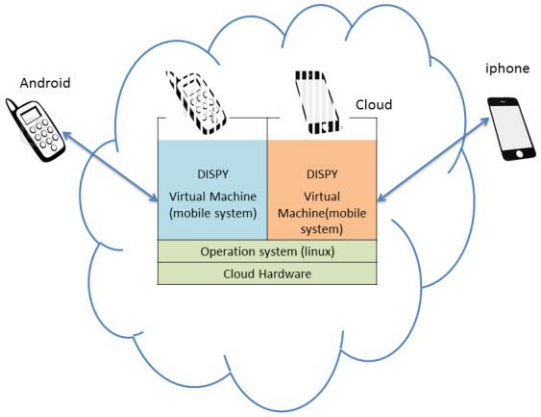

Figure 1 shows the architecture of MMPC.

B. DISPY

DISPY is used to carry out the distributed computing paradigm. DISPY is well suited for data parallel (SIMD) paradigm where a computation is conducted with different (large) datasets independently.

DISPY is implemented with asyncore [7], an independent framework for asynchronous, concurrent, distributed programming with coroutines (without threads) [8]. asyncore uses non-blocking sockets with I/O notification mechanisms epoll [9], kqueue [10] and poll, and Windows I/O Completion Ports (IOCP) for high performance and scalability, so DISPY works efficiently with a single node or large cluster(s) of nodes. After each execution is finished, the results of execution, output, errors and exception trace are made available for further processing. Nodes may become available dynamically: DISPY will schedule jobs whenever a node is available and computations can use that node.

C. Application partitioning

The algorithm requires developers to decide which part of code should be executed on the mobile phone. Developers need to mark methods as removable or non-removable at beginning. Removable methods can be offloaded to the server. Non-removable methods are:
(1) Methods that implement an application's user interface. For instance, deploying screen based methods will reduce the user experience of the application. Secondly, a display component requires large amount of data transmission. This phenomenon can reduce performance and increase energy consumption[11]. (2) Methods that interact with I/O devices. For example, a method requires a GPS module. These kinds of methods do not make sense without phone [12]. (3) Methods that interact with external components. Such as a method to access a SD card [4]. We use annotation to mark unmovable method. The reason of annotation is that: (1) Annotation can reduce the complexity of the partitioning stage. If we put more methods on the partition decision maker, then the partitioning mechanism will have more data and more conditions to consider. The obvious thing is that those methods have to execute locally. (2) Non-removable methods consume more energy if we deploy them remotely. This is because more data interaction between server and mobile phone will occur. It reduces user experience and application performance. Removable methods mean they can execute locally or remotely. Based on a particular environment, partition mechanism will decide whether to execute locally or remotely. Removable methods are marked as removable.

D. Handling failures

Failure handling is provided to detect internet disconnect or servers shut down. A time-out mechanism is instrumented to detect failures. A proxy is used to receive control back from the server. When a smart-phone loses contact with the server, then the proxy can re-invoke the method locally or it can attempt to find a spare server to invoke again. It does not affect the program's correctness, because program state is only transferred at the start and end of methods, re-executing a part of code is acceptable [13].

E. Profiler

At execution, MMPC automatically decides how to partition the application. How many methods run locally and how many methods execute remotely. Partition decisions are dependent on the three factors as below [10, 4, 2]:
(1) A smartphone’s energy consumption characteristics. If a method's energy consumption is less than the cost of transferring this method from mobile to Cloud, and will therefore increase the energy consumption. (2) Methods requirements, such as some methods need more runtime or resources. For instance, when delivering a data intensive method sacrifices internet bandwidth and makes the application unstable. (3) Network characteristics. Bandwidth, latency and packet loss are very important for this architecture. Firstly, a wireless bandwidth is always varying. A new bandwidth will cause the system to generate a new partition scheme to re-deploy the application between a mobile and servers

The profiler system monitors continuously the network and program characteristics because the network and method invocation is changing all the time. When it is decided to execute a method remotely, the code of a method and all other data which it requires will be sent to remote servers. DISPY will automatically serialize data and select

an available server to execute. After execution, all results will be sent back to the mobile phone and combined.

1) Energy Prediction

Several existing power profiling strategies exist, such as quantifying the total energy used by an application with varying throughput [14]. These techniques focus on architecture and system layer optimization rather than the application itself. The problem is that these kinds of technologies do not provide enough information for the designer to take full advantage of the application. They do not reveal how much energy it split across different parts of the application. This means we cannot trade-off between performance and energy cost with those technologies. Profiling at the application layer is typically limited to manual methods that evaluate specific design modifications proposed in a given problem context.

Several approaches can be used to measure application energy usage. The first one is to estimate energy consumption based purely on analytic models. For instance, measuring energy consumption using the relationship between algorithms' algorithmic time and space complexity to CPU or memory power model [15]. Although the accuracy needs to be improved, it is very useful for making early stage decision. The usage of this approach is limited due to the effect of various system layer optimizations, memory hierarchies and the use of low power states.

The second method is to use power models for simulating the application execution instead of analytic estimation[16]. For example, we can use a hardware power to predict energy consumption. Applications are set up to collect the power state transitions of different resources such as CPU and network. The recorded data is used to estimate energy consumption of the application. This technology suffices when there is only one application executing on a system. It is not sufficient when there is more than one application using the same resource.

The third one is to implement each resource in hardware to measure energy usage. For instance, using hardware equipment in an embedded platform to measure energy usage by each application or hardware resource [17]. The feasibility of this method is hard to predict and will cost too much.

To calculate the energy consumption based on a method, we provide a per-device solution in which the work combines the solutions of the first method and second method described above. The reason for predicting energy consumption is to decide whether this method is worth executing remotely. So, it does not need the precise energy consumption of a method. In this experiment, Firstly, we used an API provided by Android system. It can provide coarse-grained energy monitoring, which can detect energy consumption based on 1% precision of the entire phone. Then, we developed a set of benchmark tools to record CPU utilization and corresponding energy consumption. Next a simple linear model is created based on these two set of data by using least-squares linear regression. Then, we measure CPU cycles of a particular method. CPU cycles and CPU utilization can be used to make another linear model. These two linear data models can be used to predict the energy consumption of a particular method. We only need to measure how many CPU cycles to a method to estimate energy consumption. After the experiment, our result shows the median error produced by the model is about 7%, and the mean error is about 11%. This is precise enough for us to make a partition decision.

2) Program Profiling

Initially, a profiler collects three different sets of data from each method which are a methods' runtime duration, the number of CPU cycles and the state transfer requirements. The state transfer requirements include the size of methods, all relevant data which it needs during the execution and return data. A method's duration and CPU cycles are used to estimate the energy consumption of a method.

To generate the state transfer requirements, we need to calculate how much data it requires and how often this data occurs. This is because when a method is located remotely, some of the required data must invoke from a mobile phone. It is very difficult to detect the frequency of this data. Applications are not deterministic. The frequency is changing during the runtime period. Instead, we use past method invocation to predict future invocations. It is found that this idea works well for a mobile application with this architecture.

The state transfer requirements could be calculated as:

$$R = \sum_{(i,j)\in V} \{ | R_i - R_j | n(Din_{ij} + Dout_{ij}) + code_i \} \frac{1}{B * T_{tatal}}$$

D_{in} is the amount of data transferred from vertex i to vertex j and D_{out} means the amount of data sent from vertex j to vertex i. n is a number indicating how many times this data transaction occurring of total execution time. The B represents a current capacity of the bandwidth. code_{i} represents the code size of vertex i. Vertex i and j represents methods who have data interaction between each other's. T_{total} represents total transferring time of data and code. R_{i} is an indicator, when method i locates locally, then, R_{i}=0. When R_{i} located remotely, then R_{i}=1.

F. Partition Algorithm

Profiler collects variables from smartphones and treats them as a global optimization problem. In order to make the system execute efficiently, deciding how to partition the removable methods between a cell phone and servers is difficult. The system needs to find a trade-off strategy which

keeps the energy consumption to a minimum and subject to latency constraint.

Deciding where to run each function is challenging because it needs a global view of the application's behavior. The energy consumed by transferring methods remotely must be smaller than the energy consumed by executing locally. So the decision must be made globally to consider the entire program behavior.

To formulate the global optimization issue, the program is modeled as an annotated call graph. The example can be seen in Figure 2. Through the graph we can see that each vertex represents a method, each edge represents a weight of executing a left side method remotely. To optimize the problem, the graph is treated as unidirectional graph. Based on the above variables, a weight is generated as the value of the each edge. Then we apply the partition algorithm to the graph to seek a minimum cut in the graph. After this algorithm, the part with less energy cost will stay on the mobile phone but a large part of code will transfer to servers. In this work, a Min-Cut algorithm [18] is used to seek the best cutting point in an undirected graph G with vertex set V and edge set E. Every edge e has no-negative real weight.

An example is shown in Figure 2, in which an application is assumed to consist of 6 methods. Figure 2 represents a method invocation graph. G={F,G}, F represents method in Figure 2 and F={F1,..,F6}. E represents edges in the graph and E={E1,..,E6}. Each edge represents a data exchange with a weight. Firstly, The Min-Cut algorithm selects a random point. Let's assume we pick point 5. Add F5 to a most tightly connected vertex. Then, it takes F5 as a center point and selects the most tightly connected point which has the biggest weight. F6 is selects and add to the most tightly connected vertex. After that, let pretend F5 and F6 as a single point in the graph and select most tightly connected vertex with biggest weight. So, F4 is selected and added to most tightly connected group. Then, it pretends F5, F6 and F4 as a single point. F2 is selected and added to most tightly connected group. Now, we have F1 is not in the most tightly connected group and this last step of the first min-cut phrase. The induce ordering F5, F6, F4, F3, F2, F1 of the vertex. The first cut of the phrase corresponds to the partition {1}, {2, 3, 4, 5, 6} with weight w = 5. Then, min-cut algorithm executes the cut-of phrase again. Before that, F1 is merged together with F2. Figure 3 shows after the second min cut phrase (G, w, F5) and the order is F5, F6, F4, F3 and F(1,2). The second cut-of-the phrase corresponds to the partition {1, 2}, {3, 4, 5, 6} of V with the weight w =5. Figure 4 Figure 4 shows the result After the third min cut phrase, the corresponding partition is {1,3 },{2,4,5,6 }. The weight w is 4 Figure 5 shows a result after the fourth min cut phrase, the partition is {4,6 } , {1,2,3,5 }, the weight w is 5. Figure 56shows after the fifth min cut phrase, the corresponding partition is {5}, {1, 2, 3, 4, and 6} and the weight is 8.5. Overall, the minimum cut of the graph G is the third cut and the weight is 4.

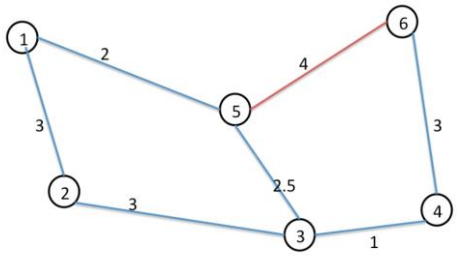

Figure 2 shows a method invocation graph.

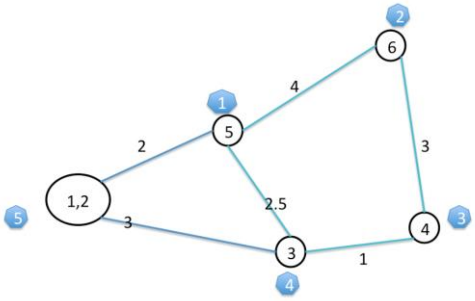

Figure 3 shows after the second min cut phrase(G, w, F5) and the order is F5, F6, F4, F3 and F(1,2). The second cut-of-the phrase correspond to the partition {1, 2}, {3, 4, 5, 6} of V with the weight w =5.

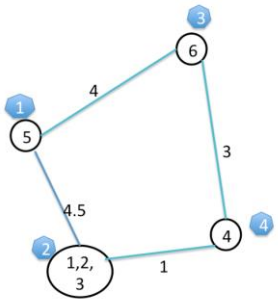

Figure 4 shows the result of third min cut phrase, the corresponding partition is {1,3 },{2,4,5,6 }. The weight w is 4

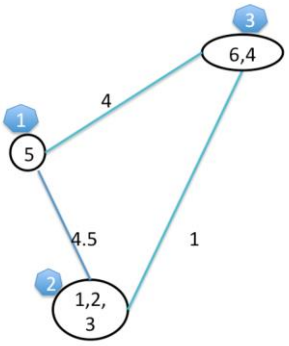

Figure 5 shows the result of fourth min cut phrase, partition is {4, 6 }, {1,2,3,5 }, the weight w is 5.

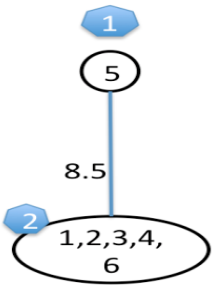

Figure 6 shows result of fifth min cut phrase, the corresponding partition is { 5},{1,2,3,4,6 } and the weight is 8.5.

III. EXPERIMENTAL SETUP

The goal of this work is to reduce energy consumption of mobile applications. In addition to saving energy, this structure can also improve the performance of applications. How much this prototype can improve the performance is very important to some sensitive applications. A mobile application, which is called power tutor (lee2012smart) is used to monitor the energy consumption based on particular applications. All results related to an application's energy consumption is detected by this application. Mobile phones normally have about 100 MB memories. This means some applications cannot be deployed onto mobile devices. A testing issue could be whether this prototype can support resource-intensive applications or not.

A. *Experiment 1*

To answer the questions above, we used three Android phone based applications. Two of them were already developed and they were open source. They are a video game application and a translator application. We deployed a Caesar encryption application which it is used to run as a data intensive application.

Three different scenarios are considered. In the first scenario, three applications are running standalone on the mobile phone. In the 2nd scenarios, MMPC is used to offload methods to servers over a Wi-Fi link with different RTT values (25ms, 50ms, and 100ms). RTT represents Round-trip time which is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received.

B. *Experiment 2*

This experiment is designed to test the performance of the partition algorithm. In order to prove that a min-cut algorithm is the best solution we deployed two more algorithms that can be used to solve the same problem, which are an exhausting partition algorithm and a linear program. The exhausting algorithm compares all possible partition schemes, and then selects the best one. The linear program collects all data from profiling process to find a most balanced partition strategy.

In this evaluation, we firstly executed three applications standalone and recorded how long it takes for execution. Then, we deployed a new prototype to offload code to servers with a link. During the execution, we inserted various queuing delay (25ms, 50ms, 100ms). Performance results are recorded for each run.

C. *Methodology and Environmental Setup*

During the evaluation, a ZET San Francisco ruing Android 2.3 with DISPY framwork 3.3 has been used. For the servers, three dual-core desktops are used with installed android simulators. The server is configured with an "tc" command that inserts packet queuing delays to control the RTT of the path between the smartphone. On the server side, Eucalyptus [19] is used to organize three computers.

IV. EVALUATION

Figure 7, 8 and 9 represent comparisons of energy consumption. The three applications were evaluated in four scenarios. A battery monitor program is used to calculate the energy consumption. Firstly, the three applications are executed on mobile phone locally. Then, four different round trip time (RTT) were inserted into the network between a cellphone and servers. Through the above figures, significant energy saving are seen in the translator and the Caesar applications. With the increase RTT, the system saves less energy. The video game application shows less drastic in energy saving but it remains non-trivial.

The RTT is added by using a Linux kernel function. Within the current distributions of Linux there is a kernel component called netem [20], which adds Network Emulation and is used for testing and simulating the same types of issues one would see in a WAN (Wide Area Network). 'tc' is a command that allows one to add rules to netem.

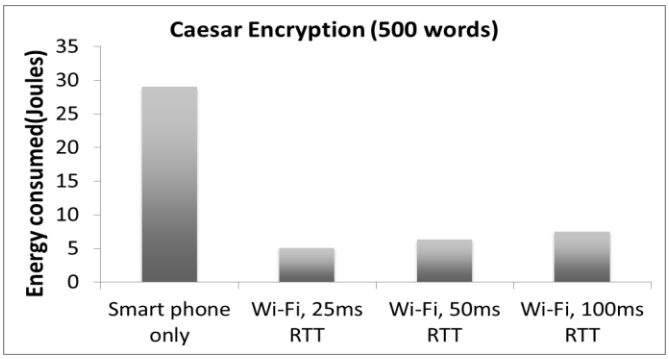

Figure 7 shows energy consumption of Caesar encryption (500 words)

e

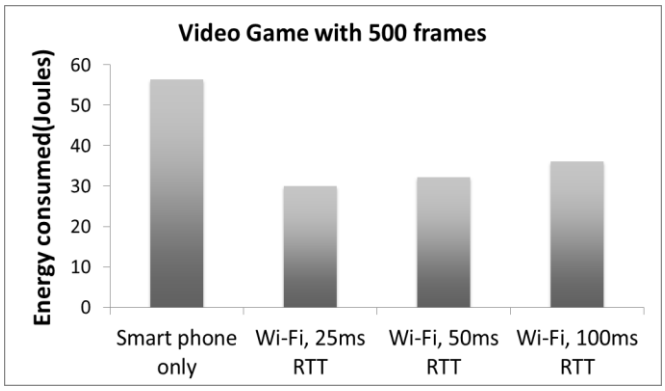

Figure 8 shows energy consumption of Video game with 500 frames

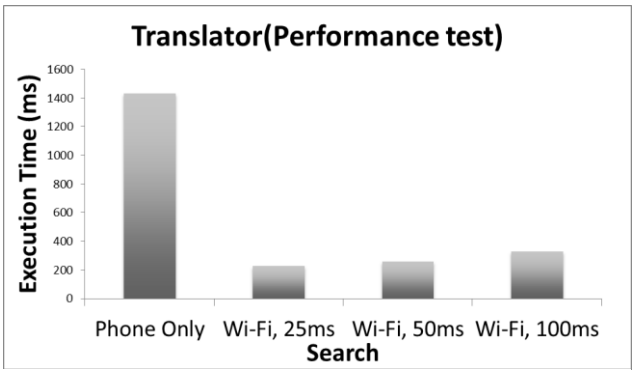

Figure 10 shows total execution time with different conditions

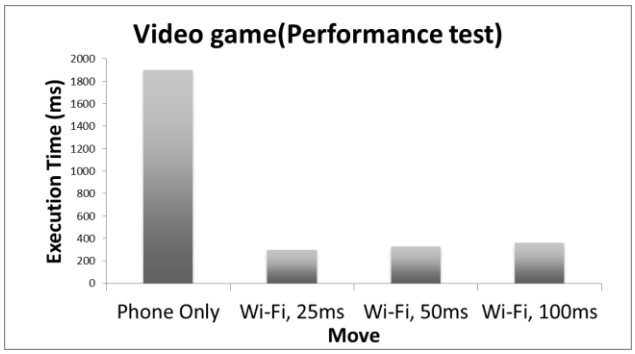

Figure 12 total execution time ran by 4 scenarios and a move method was delivered to Cloud (smart phone only).

Figure 10 shows the comparison of three different partition algorithms. The experiments test the runtime duration of each algorithm. 1000 different partition schemes are created to find the best one. The results show that the exhausting partition scheme consumes much more energy than the other two algorithms. The min-Cut algorithm consumes least energy.

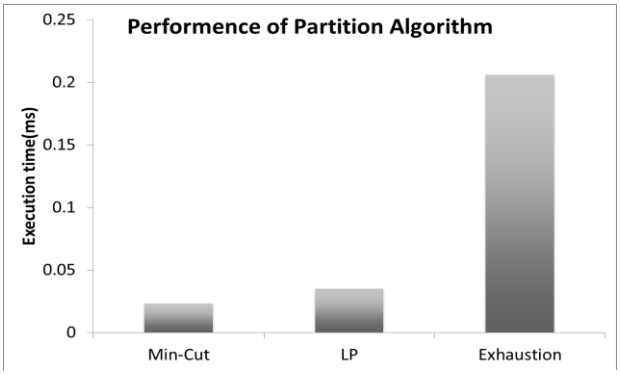

Figure 9 shows that three different partition algorithms were evaluated and compared.

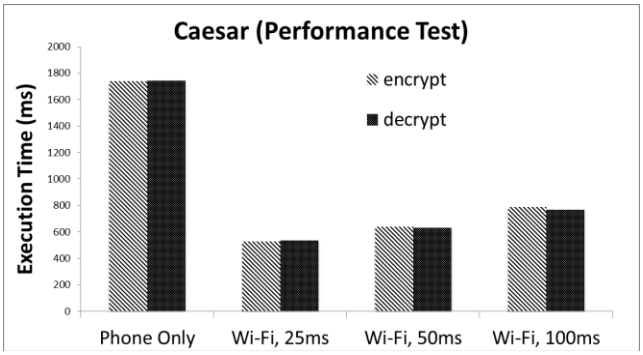

Figure 13 shows the experiment was run by 4 scenarios and decrypt and encrypt methods were delivered to Cloud (smartphone only).

Through Figure 11 and 12, the performance is generally improved although it degrades when the latency is very big (> 100 ms). This is probably because of the chatty nature of the handshaking protocol, which makes Wi-Fi's high power state last longer.

Figure 13 describes the performance of the Caesar application in four different scenarios. The Caesar application is designed for the testing of remote execution. The application consists of three methods: a data generator,

an encrypt method and a decrypt methods. Firstly, we executed the entire application on a mobile phone. Then, we transported the encrypt and decrypt methods to the cloud for remote execution. Encrypt and decrypt should take the same amount of time and the small difference is probably because of the monitor software.

A resource-intensive application is tested on the cloud as it cannot be run on a mobile phone due to its high resource requirements. During this experiment, the Caesar application was used to process a very large dataset. Figure 14 shows the memory consumption of the Caesar application, which is around 80MB. The mobile phone's RAM is about 100 MB, which implies that such a big application cannot be run on the mobile phone.

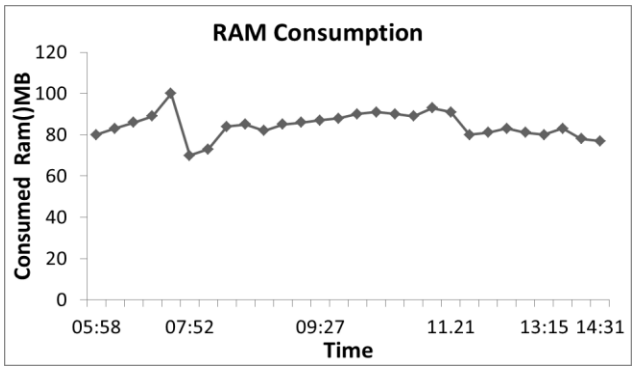

Figure 14 shows Memory consumption of the Caesar application (smartphone only).

V. CONCLUSIONS

In this paper, we have implemented MMPC (Mirroring Mobile Phone in the Clouds) and there are two original contributions in implementing MMPC: 1. a dynamic program profiler to analyze the program behaviors. The profiler measures initially the behaviors of the Internet and the mentioned methods and then continually monitors the program behaviors. Such a continuous monitoring is crucial as the old information may lead to wrong decisions when offloading. 2. Dynamic Program Partitioning Algorithm to divide a mobile application program into two pieces, the small one of which remains in the mobile phone whereas the large one goes to the Cloud to be executed there. Because the Cloud represents an enormous execution power and storage space so the overall performance of executing a (large) mobile application is much improved.

REFERENCE

[1] K. Lahiri, S. Dey, D. Panigrahi, and A. Raghunathan, "Battery-driven system design: A new frontier in low power design," in *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, 2002, p. 261.

[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 *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[3] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer (Long. Beach. Calif).*, vol. 43, no. 4, pp. 51–56, 2010.

[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[5] G. Pemmasani, "dispy : Python Framework for Distributed and Parallel Computing," 2013.

[6] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wirel. Commun. Mob. Comput.*, 2011.

[7] M. L. Hetland, "Project 5: A Virtual Tea Party," *Begin. Python From Novice to Prof.*, pp. 455–472, 2005.

[8] G. Kahn, D. MacQueen, and others, "Coroutines and networks of parallel processes," 1976.

[9] P. Verhaeghe, K. Verslype, J. Lapon, V. Naessens, and B. De Decker, "A mobile and reliable anonymous epoll infrastructure," in *Security and Privacy in Mobile Information and Communication Systems*, Springer, 2010, pp. 41–52.

[10] J. Lemon, "Kqueue-A Generic and Scalable Event Notification Facility.," in *USENIX Annual Technical Conference, FREENIX Track*, 2001, pp. 141–153.

[11] S. Abolfazli, Z. Sanaei, and A. Gani, "Mobile cloud computing: A review on smartphone augmentation approaches," *arXiv Prepr. arXiv1205.0451*, 2012.

[12] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010, p. 6.

[13] J. Flinn, "Cyber foraging: Bridging mobile and cloud computing," *Synth. Lect. Mob. Pervasive Comput.*, vol. 7, no. 2, pp. 1–103, 2012.

[14] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis, “JouleSort: a balanced energy-efficiency benchmark,” in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 365–376.

[15] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, “Energy consumption in mobile phones: a measurement study and implications for network applications,” in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009, pp. 280–293.

[16] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 105–114.

[17] D. McIntire, T. Stathopoulos, and W. Kaiser, “etop-Sensor Network Application Energy Profiling on the LEAP2 Platform,” in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, 2007, pp. 576–577.

[18] M. Stoer and F. Wagner, “A simple min-cut algorithm,” *J. ACM*, vol. 44, no. 4, pp. 585–591, 1997.

[19] K. G. Eldridge, J. Davidson, C. E. Harwood, G. van Wyk, and others, *Eucalypt domestication and breeding*. Clarendon Press, 1993.

[20] S. Hemminger and others, “Network emulation with NetEm,” in *Linux Conf Au*, 2005, pp. 18–23.