Towards Automating Mobile Cloud Computing Offloading Decisions: An Experimental Approach

Roberto Beraldi, Khalil Massri
Computer and System Science Department
La Sapienza University of Rome
Rome, Italy
{beraldi, massri}@dis.uniroma1.it

Abderrahmen Mtibaa, Hussein Alnuweiri
Department of Electrical and Computer Engineering
Texas A&M University
Doha, Qatar
{amtibaa, alnuweiri}@tamu.edu

***Abstract*—Mobile applications require more and more resources to be able to execute tasks on a single device, despite the fact that mobile devices are getting better capabilities. This has been addressed through several proposals for efficient computation offloading from mobile devices to remote cloud resources or closely located computing resources known as cloudlets. In this paper we adopt a experimental driven approach to highlight the offloading tradeoffs. We show that rather than always offloading tasks to a remote machine, running particular tasks locally can be more advantageous. We propose a novel generic architecture that can be integrated to any mobile cloud computing application in order to automate the offloading decision and help these applications to improve their response time while minimizing the overall energy consumed by the mobile device.**

***Index Terms*—Mobile Cloud Computing, Chess Game, Android Experimentation.**

I. Introduction

Mobile devices are increasingly utilized beyond simple connectivity for services that require more complex processing and capabilities. These include pattern recognition to aid in identifying snippets of audio or recognizing images, reality augmentation to enhance our daily lives, collaborative applications that enhance distributed decision making, planning and coordination. These applications are already in ubiquitous use today, others are still prototypes awaiting the next generational change in device capability and connectivity.

Cloud computing, in general, is reshaping the design and implementation of today's software applications. These applications are designed originally for desktops that are always connected to the Internet. While traditional cloud applications has been quite successful, to-date it suffers from a number of shortcomings especially with the presence of wireless communication at the edge. Shortcomings include the high latency and energy consumption caused by the intermittent aspect of wireless networks, which makes executing tasks locally more advantageous in certain circumstances.

In this paper, we adopt an experimental based approach in order to highlight the need of an automated offloading mechanism which decide whether a given task should run locally or remotely at the cloud. We propose a generic middleware architecture that can be plugged into any mobile cloud computing (MCC) application. For a specific task, based on the task characteristics and device capabilities, our architecture decides whether the task should be offloaded to a distant cloud or run locally on the device itself. We then implement a chess game as prototyping mobile application in order to identify under which circumstances would migrating be advantageous. We used the chess game as a real testbed environment to identify all the factors that help make an efficient offloading decision with respect to users' preferences or minimizing the overall resources usage.

The rest of this paper is organized as follows. Section 2 briefly discusses work related to our research. In Section 3 we describe the design of our generic architecture for mobile cloud computing applications. Section 4 describes our experimental platform. Section 5 presents the results from an experimental evaluation. Section 6 summarizes our findings and discusses our future work agenda in this area.

II. Related Work

Leveraging mobile networking and cloud computing attracts many researchers nowadays [4]. [6] was one of the earliest solutions for dynamic partitioning among mobile computers and a fixed infrastructure. There are a number of offloading frameworks that can support the development of offloadable applications. Offloading can be achieved at the level of services, methods and system. Service offloading intercepts those parts of the code that a software developer has manually set up for offloading. Cuckoo [5] integrates into Android applications by creating a proxy inside the application for the interfaces that the application developer has defined. The proxy then decides whether to invoke its corresponding local method or to migrate the computation to the surrogate. Method offloading, however, uses per-method annotations and wraps methods directly for proxying. This approach is less intrusive from application developer's viewpoint in the sense that it does not conceptually require strict separation of offloadable code parts. MAUI [3] implements this ideology. It investigates the energy consumption challenge when offloading computationally heavy tasks to a cloud rather than executing locally. MAUI relies on developer effort to convert mobile applications to support such decision making, and secondly, it only considers the possibility of offloading to different types of infrastructures. CloneCloud [2] presents a solution which decides whether to execute a task on a remote cloud

 ISBN: 978-1-61208-305-6

service versus executing it locally based on static analysis and dynamic profiling information of a task. CloneCloud, on the other hand, uses a modified virtual machine implementation of Android to intercept running threads at byte-code level and to migrate them for distributed concurrency. As a side effect in reducing burden to the application developer, image-level offloading frameworks are required to be more sophisticated.

In fact, with the advent of mobile device capabilities, migrating task computation always to powerful machine is questionable. We believe that running certain tasks locally on mobile devices can be more advantageous and may save both energy and time especially in the presence of intermittent wireless connectivity. In this paper, we adopt an experimental approach towards identifying the potential gain of mobile computational offloading in regards of both response time and energy consumption, and propose a design of a novel generic architecture that help actual application to make the best offloading decision based on these metrics.

III. Mobile Cloud Computing Architecture: Migrate vs. Run Locally

Mobile cloud computing is indeed becoming a dominant trend. However, current systems mainly focus on (i) offloading all functionalities to the cloud via simple client server architectures, or (ii) implementing applications and services that run locally on the mobile device. In this paper, we propose a novel architecture that leverages the two previous cases and computes, in runtime, the best offloading method with regards to two main metrics: the total response time and the overall energy consumed by the mobile device.

As summarized by Figure 1-(a), our architecture proposes a generic middle-ware that can be plugged into any mobile cloud computing application. Such architecture receives a given task T from the mobile computing application and based on the device capabilities (*e.g.,* CPU usage, memory, available energy), it computes an utility function which helps deciding whether the task T should be offloaded to a distant cloud or run locally on the device itself.

Task Modeling Engine: this engine is responsible of receiving tasks from the application. It models each task T by a combination of data, D_T, taken as input to perform such a task, and computation, C_T, that the task needs to perform on this data in order to yield a result. A given application consists of many tasks, and the more data and computation intensive these tasks are, the more time and energy required to perform them.

Decision Maker: it is the main engine of our architecture. It uses data from the device data base and cache and triggers the estimators engines in order to make a final decision about the offloading method of the given task T. It receives T from the task modeling engines and computes an utility function in order to send back the task to the application for local execution, or forward it the task forwarder for remote execution at the cloud.

Energy and Delay Estimators: the energy and delay estimators are responsible of estimating (i) the approximate energy to be consumed after running the task locally or remotely at the cloud, and (ii) the total response time $t_T = t_T^{end} - t_T^{beg}$, where t_T^{end} and t_T^{end} represent respectively the time in which the application receives the results of the task T, and the time in which the application sent the task to the task modeling engine. The estimators take as input, the task characteristics which are D_T, and C_T in addition to historical decisions of similar tasks (stored in the task/decision cache)

Task Forwarder: Upon receiving an order to migrate the task to distant cloud via the decision maker, the forwarder forwards the task to the distant cloud and keeps track about the status of the connection. In case of intermittent connectivity the forwarder reports an additional delay to the decision maker which will remake the decision based on the new factors (additional estimated delay).

Databases and Cache: the databases store the device capabilities, the communication technologies (*e.g.,* wireless, 3G, 4G, Bluetooth) and the task/decision cache. The cache is implemented to avoid remaking decisions for similar or identical tasks.

This architecture is built on the promise of computational offloading gain experienced, either in energy saving or task completion time, by any given cloud. If no potential gain/advantage exists, there is no point in adopting this architecture in the first place. Consequently, we believe that the major problem that needs to be addressed at this early stage of research is answering the question of when should we offload a given task. Therefore, we focus our attention in the remainder of this paper to quantitatively verify the potential gain tradeoff between energy and time while executing the task locally or offloaded remotely.

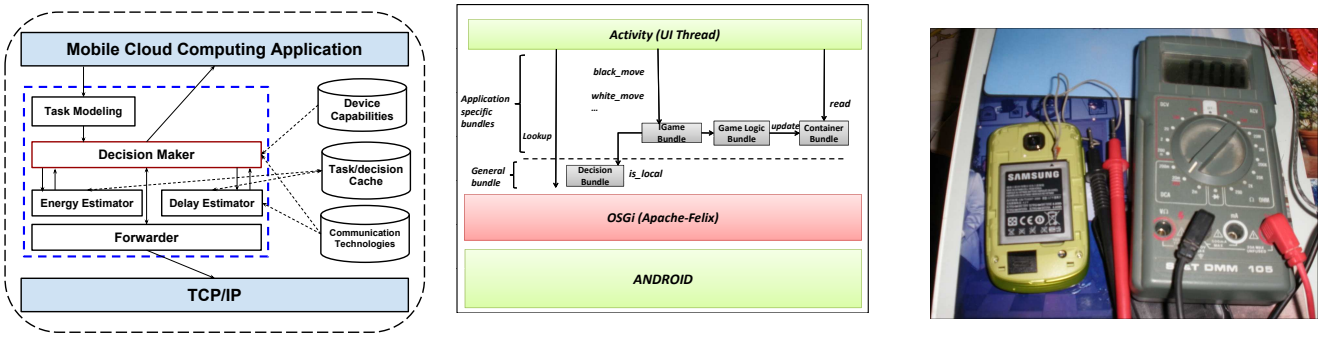
IV. Experimental Testbed: Chess Game Prototype

Our goal consists of making "good" decision about migrating computation of a given task or running it locally. It involves making a decision regarding which task is worth offloading. In order to answer this question, we adopt an experimental approach using a real testbed environment. Our goal is mainly to: (i) identify the trade-off between the gain of migrating tasks as opposed to running them locally, and (ii) identify under which circumstances would migrating be advantageous.

We choose to implement a chess game as it has a single task whose complexity can be easily set according to the expertise level, *e.g.,* from beginner to expert.

Our application is divided into three major software layers as shown in Figure 1-(b). The bottom layer is the standard OSGi's implementation of the Apache Software foundation community, Felix [1]. This software layer basically allows to register and lookup for other OSGi bundles. We also implement the following four bundles; (1) The *decision* bundle that encapsulates the decision logic about running an application module locally, i.e., in the device, or remotely in the cloud. It roughly corresponds to the Decision Maker component of Figure 1. (2) The `IGame` bundle is the interface seen by the

 ISBN: 978-1-61208-305-6



(a) Our high level architecture (b) Prototyping implementation (c) Energy testbed

Fig. 1. System architecture and experimental testbed

android application. This bundle implements the following operations: initBoard, initSearch, blackMove and whiteMove. (3) The `GameLogic` bundle implements the computer move and occupies 456 KB. This is the bundle (Task) to be uploaded into the cloud for implementing remote calls. (4) The `container` bundle is used as access point to read the data generated after a move.

A game tree for the chess game is used to decide the best counter move. It is composed of nodes representing the state of the board and edges the possible moves. The tree is explored at different levels, according to the difficulty degree, labeled from 1 (beginner) to 4 (expert). The maximum explored level of the tree was, respectively, 3, 7, 10, 14. The best counter move is searched applying the alpha-beta searching algorithm. This algorithm exploits lower and upper bounds (named alpha and beta), to prune part of the game tree that cannot possibly influence the final decision.

As far as the cloud technology is concerned, we have used OpenShift, a Red Hat's free, auto-scaling Platform as a Service (PaaS) for applications. It allows to run different VM, called gears. It uses the notion of cartridge, a set of predefined technologies that can be installed and run inside gears. In addition, Openshift allows users to exploit the powerful Do-It-Yourself (DIY) feature that allows for running just about any program that speaks HTTP. As the Apache felix is not available as a cartridge we have used the DIY feature for our experiment. Moreover, in order to allow android openshift communication we have exploited Apache CXF, an open source services framework that helps to build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. In particular we have exported the service in the cloud as RESTful HTTP endpoints . CXF allows to export OSGi services as Web Services.

V. Experimental Analysis

We have tested the application in three different settings, named local, cloud and cloudlet. In the local mode, the mobile device executes all the code, whereas in the cloud and cloudlet modes, the device only executes the code required for UI updating and the user moves. In the cloudlet node, the remote task is executed in a VM hosted by a server machine located in the same wireless LAN of the device.

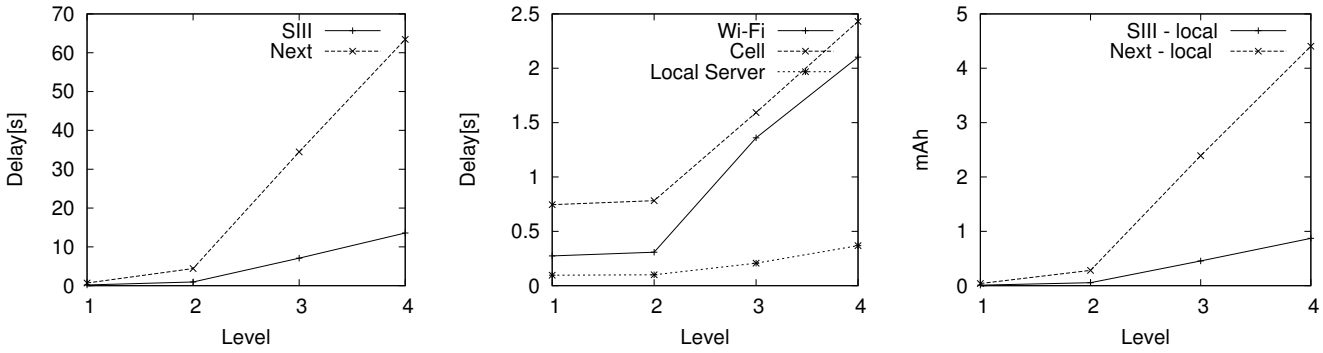
The opening phase in a chess game is a very important phase, as it may shape the way the whole match will proceed. The reply to an opening is then a good situation game to test the performance of our application. We have considered a famous openings due to Anderssen, A2 in A3. Two different mobile devices have been used in the experiments; Samsung Galaxy Next Turbo and a Galaxy SIII. In the cloud mode, the game logic runs in the openshift platform and Internet is accessed either through a wi-fi or via GSM. Finally, in the cloudlet setting, the same public cloud environment runs locally on a PC running Windows 7 Home Premium, CPU i7-3610QM CPU @2.30GHz with 8 GB RAM.

Response Time: The time elapsed from when the white ends to move till the black move ends is called the response time. To measure such a response time the same opening was repeated three times. Figure 2 shows the average response time for (a) local execution and (b) remote execution. Under the local execution case, it is clear that as the complexity of the task increases, as in level 3 and 4, the response time increases considerably, especially, for the lower computation capability one (Next). While, on the other hand, when the task execution is done remotely, either on the cloud or cloudlet, the response time is always acceptable, even when the task complexity is the maximum. The level can thus be used both to describe the task complexity and to drive decision maker to select the execute mode. In addition it also affects the power consumption, as we discuss in the following subsection.

Energy Consumed Locally: Energy saving is one of the most important expected benefit that mobile cloud computing should provide. Figure 2-c reports the average mAh for the local execution mode. This plot is similar to the delay's one. This is due to the fact that the delay to the reply is indeed due to the computation of the black move. In other words, if the computation requires T s then the required charge is kT, where k is a constant independent from T.

Quantifying the Overhead: In the previous section, we were assuming that the cloud implements already chess game algorithms and maintains a state about each player of a game. In this section, we investigate a scenario where the application is totally implemented in the phone device itself and a remote execution of a task requires transferring a chunk of data that is needed to run the tasks remotely.

 ISBN: 978-1-61208-305-6



(a) Response time for local execution (b) Response time for remote execution (c) Energy vs. difficulty levels

Fig. 2. Response time and energy performance of local and remote execution

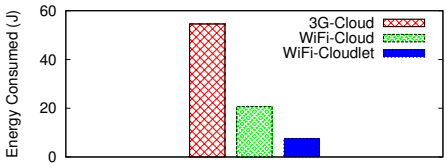


Fig. 3. Energy consumed while offloading to cloud or cloudlet using SIII

We have measured the number of bytes exchanged between the client and cloud during the initialization phase (when bundles are emitted to the cloud) and during moves. For this purpose we have used the wire shark sniffer. We then run a set of experiments to quantify the energy overhead related to communication between the mobile device and the cloud as shown in Figure 1-(c). We remove the battery of the Samsung SIII device and solder wires coming from a power supply into the battery contacts of the device. The power supply that we use comes with a built in ammeter and voltmeter. We then provide a constant voltage according to the manufacturer specifications and power the device on. Using the current and voltage readings from the ammeter and voltmeter respectively, we are able to determine the power being consumed by the phone at any instance.

We run each experiment 5 times and compute the energy readings while idle, and those while making wireless transfers (sending or receiving data). Figure 3 compares the energy consumed while transmitting the initial code to a distant cloud or a nearby cloudlet. We consider 3G and WiFi as two wireless technologies to communicate with a distant cloud and WiFi to communicate with cloudlet.

Figure 3 shows that 3G consumes more than double the energy required to communicate using WiFi. This may imply running tasks locally if only 3G connectivity is available to reach a distant cloud. Using WiFi, communicating with a nearby cloudlet is 2 to 3 times less expensive than communicating to a cloud. This confirms the results showing in [7]. However, the cloudlets are less computationally powerful than a cloud which may introduce additional delay and therefore energy to finish executing the task.

VI. Summary & Discussion

We believe that there is a need for a generic and flexible framework that can be integrated by any mobile cloud computing to automatically decide based on a given characteristic of the application or its tasks whether it is better to run execution locally or remotely. We begin by running a set on preliminary experiments in order to identify a tradeoff between two conflicting metrics. Our preliminary experimental results have shown that energy and time depend on the computation complexity of the task and the device capabilities. On the other hand, when offloading the task to the cloud the energy and time consumed will highly depends on the communication technology used. In our work, we have adopted a first step towards automating an offloading decision maker, which measures the task complexity, then based on the device capability and the communication heuristic profile, it can choose the better choice. We plan to expand our experimental and analytical results to help identify the objection function that will be used by our decision maker. The objection function main goal is to quantify the gain based on user preferences or simply minimizing the overall resource usage.

Acknowledgment

This research was supported by a research grant from the Qatar National Research Fund under project NPRP 09-1116-1172.

References

[1] Apache felix. http://felix.apache.org/.

[2] 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*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[3] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *MobiSys'10*, pages 49–62, 2010.

[4] J. Flinn. Cyber foraging: Bridging mobile and cloud computing. *Synthesis Lectures on Mobile and Pervasive Computing*, 7(2):1–103, 2012.

[5] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: A computation offloading framework for smartphones. pages 59–79. Springer Berlin Heidelberg, 2012.

[6] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):19–26, Jan. 1998.

[7] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

 ISBN: 978-1-61208-305-6