Adaptive Code Offloading for Mobile Cloud Applications: Exploiting Fuzzy Sets and Evidence-based Learning

Huber Flores
huber@ut.ee

Satish Narayana Srirama
srirama@ut.ee

Institute of Computer Science, Mobile Cloud Lab
University of Tartu
J. Liivi 2, Tartu, Estonia

ABSTRACT

Mobile cloud computing is arising as a prominent domain that is seeking to bring the massive advantages of the cloud to the resource constrained smartphones, by following a delegation or offloading criteria. In a delegation model, a mobile device consumes services from multiple clouds by efficiently utilizing solutions like middleware. In the offloading model, a mobile application is partitioned and analyzed so that the most computational expensive operations at code level can be identified and offloaded for remote processing. While code offloading is studied extensively for the development of mobile cloud applications, much of the advantages of cloud computing are still left unexploited and poorly considered in these approaches. Cloud computing may introduce many other dynamic variables like performance metrics, parallelization of tasks, elasticity etc., to current code offloading models that could affect the overall offloading decision process. To address this, we propose a fuzzy decision engine for code offloading, that considers both mobile and cloud variables. The cloud parameters and rules are introduced asynchronously to the mobile, using notification services. The paper also proposes a strategy to enrich the offloading decision process with evidence-based learning methods, by exploiting cloud processing capabilities over code offloading traces.

Categories and Subject Descriptors

C.1.3 [**Computer Systems Organization**]: Other Architecture Styles—*Cellular architecture (e.g., mobile)*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/Server*

General Terms

Design, Reliability

MCS'13, June 25, 2013, Taipei, Taiwan

Keywords

Mobile Cloud Computing, Code Offloading, Machine Learning, Fuzzy Logic

1. INTRODUCTION

Cloud computing [2] becomes mobile when a handset tries to access the shared pool of computing resources provided by the cloud, on demand [9]. Mobile cloud applications [10, 7] are considered as the next generation of mobile applications, due to their promise of linked and elastic computational cloud functionality that enables to augment their processing capabilities on demand, power-aware decision mechanisms that allow to utilize efficiently the resources of the device and dynamic resource allocation approaches that allow to program and utilize cloud services at different levels (SaaS, IaaS, PaaS).

Mobile applications may be bonded to cloud resources by following a delegation or offloading criteria. In a delegation model, a mobile device utilizes the cloud to perform resource-intensive operations which are time-consuming, programmable and parallelizable among multiple servers (e.g. based on distributed frameworks like MapReduce), and computationally unfeasible for offline devices. From a delegation perspective, hybrid cloud and cloud interoperability are essential for mobile scenarios in order to foster the de-coupling of the handset to a specific cloud vendor, to enrich the mobile applications with the variety of cloud services provided on the Web and to create new business opportunities and alliances. However, developing a mobile cloud application in this model involves adapting different Web APIs from different cloud vendors within a native mobile platform. We have studied the delegation of mobile tasks to hybrid clouds in detail and have developed a Mobile Cloud Middleware [9] framework (MCM) that addresses the issues of interoperability across multiple clouds, transparent delegation and asynchronous execution of mobile tasks that require resource-intensive processing, and dynamic allocation of cloud infrastructure.

In contrast to delegation, in an offloading model, a mobile application may be partitioned (e.g. methods, classes) and analyzed *a priori* (at development stages) or *a posteriori* (at runtime) so that the most computational expensive operations at code level can be identified and offloaded for remote processing. A mobile operation may be offloaded or not, depending on the impact of its execution over the mobile resources. Conceptually, offloading is preferable only if a mobile operation requires high amounts of computational

processing and the same time, low amounts of data need to be sent in the communication. Otherwise, offloading is not encouraged [18]. Unlike delegation, offloading does not require online connectivity at all times, as the tasks can also be executed by the mobile in standalone. However, offloaded mobile operations require a low-level scale processing, when compared with delegated mobile tasks.

Recently, code offloading has re-gained a lot of interest towards the development of mobile cloud applications and most of the research works in this domain have proposed solutions to bring the cloud to the vicinity of a mobile [10, 4, 3, 13, 17] from an offloading perspective. They also address partially the issues of determining *what*, *when* and *how* to offload from mobile to cloud. Basically, these solutions foster an architecture, in which, the cloud provides the virtual computational resources and the mobile introduces the partition strategies (e.g. static analysis), the decision logic based on local parameters (e.g. network bandwidth) and the basic implementation primitives (e.g. annotations) that enable to synchronize a mobile application stack with a virtual machine running in the cloud.

However, given this context, we can argue that much of the advantages of cloud computing are left unexploited and poorly considered. A cloud does not just mean a virtual machine or a pool of servers which are accessible from the Internet. It has its own intrinsic features like elasticity, utility computing, fine-grained billing, illusion of infinite resources, parallelization of tasks, etc. So an ideal mobile cloud framework should take advantage of several of these features. Cloud computing may introduce many other dynamic variables (e.g. performance metrics) to current code offloading models that could affect the overall offloading decision process. For instance, performance metrics (e.g. CPU load) of the instance/cluster at the cloud may be useful by the mobile in order to determine 1) whether a server is not that busy so that it can handle an incoming request and 2) a dynamic execution plan that allows to parallelize mobile operations in a single machine with multiple cores or in different machines with a single core. Consequently, a code offloading model should not just target mobility aspects, but also target oscillating changes in cloud infrastructure. Moreover, we think that a mobile cloud architecture must be one that not just increases the storage and computational functionalities of the smartphones when communication is suitable, but rather, uses its inherent capabilities to process *big data* in order to enhance periodically the mechanisms of the devices with offline cloud analysis (e.g. exploration of code offloading traces) delivered asynchronously using cloud-based mechanisms such as notification services, so that each time a device may have opportunity to interact with the cloud again, the handset does it better using refined profiling strategies that allow to increase performance and save energy.

In order to investigate a strategy that enables to enhance the code offloading decision process of a mobile device with cloud power, we envisioned in this paper, a learning code offloading approach that enables to transform raw code offloading traces into knowledge that can be used by the handset to counter issues such as device diversity, adaptive application execution and unpredictable code profiling. Moreover, the strategy also allows to introduce and consider many other distributed parameters of a mobile cloud architecture. The contributions consist of:

- A decision engine that implements a fuzzy logic [16] model that considers mobile and cloud variables into the decision process. Cloud parameters and rules are introduced asynchronously from cloud to mobile using notification services. Rules are generated dynamically in the cloud by analyzing code offloading traces with a conceptual neuronal networks algorithm.
- A prototype implementation of the fuzzy logic engine.

The remainder of this paper is structured as follow: Section 2 addresses the related work. Section 3 presents the overall conceptual architecture. Section 4 describes a use case of the approach. Section 5 explains the state of our current implementation. Section 6 introduces the fuzzy logic decision model for code offloading. Section 7 presents an evaluation of the asynchronous mechanism and section 8 concludes the paper with future research directions.

2. RELATED WORK

Recently, code offloading has been re-discovered as a technique that allows to empower the computational capabilities of mobiles with cloud resources. Code offloading refers to a technique, in which resource intensive mobile components are identified and offloaded to remote processing in order to be executed by cloud-based surrogates. Most of the prominent works in the domain have proposed solutions to overcome the issues related with deciding whether to offload or not to cloud. Prominent works in this domain are described in table 1, considering two perspectives, mobile and cloud. Based on this information, we can clearly distinguish the role of the mobile and cloud in a common mobile cloud architecture. From the table, we can also observe that currently, most of the effort has been focused on providing the device with a local context logic that is utilized to offload components, dynamically, when it is beneficial for the mobile resources.

Most prominent of these works are MAUI [4] and CloneCloud [3]. MAUI proposes a strategy based on code annotations to determine which methods from a Class must be offloaded. Annotations are introduced within the source code by the developer at development phase itself. At runtime, methods are identified by the MAUI profiler, which basically performs the offloading over the methods, if bandwidth of the network and data transfer conditions are ideal. However, MAUI does not address issues of adapting the mobile application for different devices and does not make advantage of the scalability feature of the cloud. In MAUI, for every new application, one needs to build a new Server Proxy.

Similarly, CloneCloud proposes an approach based on static analysis to dynamically partition a mobile application so that it can be executed seamless in different devices. Moreover, CloneCloud also proposes the encapsulation of the mobile application's stack into a virtual machine running in the cloud. This ensures that at bytecode level, mobile operations can be processed locally or remotely, once both parties are synchronized. The decision to offload or not to offload a mobile operation is taken by a dynamic profiler similar to MAUI. CloneCloud approach is process-based, tries to extrapolate pieces of the binary code of a given process whose execution on the cloud would make the overall process execution faster.

Most of the other approaches addressed in table 1 also follow the same principles of dynamic partitioning and virtual

machine synchronization. ThinkAir [17] framework is one which is targeted at increasing the power of smartphones using cloud computing. ThinkAir addresses MAUI's lack of scalability by creating virtual machines (VMs) of a complete smartphone system on the cloud. Moreover, ThinkAir provides an efficient way to perform on-demand resource allocation, and exploits parallelism by dynamically creating, resuming, and destroying VMs in the cloud when needed. However, since the development of mobile application uses annotations, the developer must follow a brute-forced approach to adapt his/her application to a specific device. Moreover, resource allocation in the cloud seems to be static from the handset as the device must be aware of the infrastructure with anticipation. Thus, the approach is neither flexible nor fault tolerant.

COMET [13] is another framework for code offloading, which differs from the other approaches in terms of implementation. Basically, it focuses on how to offload rather on what and when. COMET's runtime system allows unmodified multi-threaded applications to use multiple machines. The system allows threads to migrate freely between machines depending on the workload. COMET is a realization built on top of the Dalvik Virtual Machine and leverages the underlying memory model of the runtime to implement distributed shared memory (DSM) with as few interactions between machines as possible. COMET makes use of VM-synchronization primitives.

Unlike previous approaches, we proposed in this work, a strategy that allows to enrich the offloading decision process of a device by exploiting cloud processing capabilities with evidence-based learning methods, over code offloading traces. Moreover, we think that offloading is not a decision process that happens just in the device, but rather offloading is a learning process that involves a global understanding of the complete mobile cloud infrastructure. We envision our strategy as a solution to overcome the issues of adaptive application partitioning, offloading decision-making and cloud-aware dynamic resource allocation. Most similar approach that can be comparable with our proposed solution is an adaptive offloading mechanism which targets pervasive devices [14]. However, this work does not involve cloud at all.

3. TOWARDS A MOBILE CLOUD CONTROL SYSTEM

In mobile code offloading, while some components of a mobile application may be marked explicitly for remote execution by a software developer [4] or implicitly by an automated mechanism [3], it is the decision engine, which is in charge of deciding whether a mobile component (aka mobile operation) is offloaded or not. Basically, a decision engine profiles multiple local aspects of the device (e.g. bandwidth connectivity, size of data, etc.) and applies certain logic over them (e.g. linear programming), so that the engine can measure whether the handset obtains or not a concrete benefit (e.g. extended battery life) from the offloading.

Mobile application partitioning is preferable by using an automated mechanism at runtime as it provides flexibility to execute the same application on multiple devices with different hardware properties, and thus avoiding a brute-force mobile development approach, which consists of developing the same application for a specific device as needed. However, mobile components are offloaded according to the local context of the device. Therefore, in some cases it may be possible that even identical devices with similar local contexts and with same sequence of offloading execution could present different offloading variations, which can be translated into a real benefit or extra overhead for the handset. This can easily be realized as cloud infrastructure parameters play an important role in the overall system. For instance, even though, the execution of an offloaded mobile component is faster in a remote cloud-based instance [13] than in a mobile, the execution time of an offloaded mobile component depends directly on the processing capabilities of the instance. In other words, the higher the computational capabilities the faster the execution of a mobile component. Notice that this behavior is not exponential and in some cases its dependable on how the code is written to take advantage of the underlying hardware.

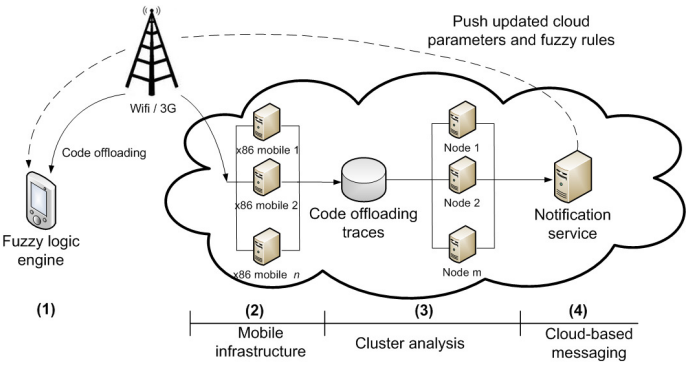

Figure 1: Evidence-based code offloading architecture

However, a global architecture understanding about what and when to offload to the cloud, may be found in the offloading information traces which are generated by the massive amount of devices that connect to the cloud. For instance, it could be possible to find which type of instance provides greater offloading benefits to a specific device. On the basis of these assumptions, we propose a mobile cloud architecture that enhances the decision offloading process of a device by introducing dynamic cloud parameters and exploring code offloading traces. The complete architecture is shown in figure 1 and consists of 1) a mobile offloading decision mechanism based on fuzzy logic, 2) a virtualized mobile infrastructure (based on Android x86), 3) a repository of code offloading traces along with a cluster to analyze it and 4) a cloud-based messaging framework to push data to the handset asynchronously.

(1) Mobile offloading logic

At the mobile platform level, the device implements a decision engine based on fuzzy logic, which is utilized to combine n number of variables, which are obtained from the overall mobile cloud architecture. Fuzzy logic is preferable for this kind of decisions as it provides more than a yes or no answer to the question *when to offload*. Basically, fuzzy logic assigns a degree of truth to an offloading decision input so that it can be analyzed based on multiple intervals and rules. The information utilized to build the intervals and to define the rules is generated by the mobile prolifers and

Framework			Mobile Perspective			Cloud Perspective		
Name	Code offloading approach	Code decision logic	Device parameters	Local implementation primitives	Remarks	Remote parameters	Remote implementation primitives	Remarks
MAUI	Code annotations	Dynamic profiler, Decision based on local parameters	Network bandwidth Data transfered Energy of method	LP model Windows Phone 7	Improved batterylife Increased performance	None	.Net framework single server	Lacks cloud scalability
CloneCloud	VM-sync at thread level	Static analysis, Dynamic profiler, Decision based on local parameters	Network bandwidth, CPU level of device, energy consumed	LP model, modified Dalvik	Performance increased up to 20x	None	Dalvik x86 single server	Limited multithreading, Blocked on shared states, uncontrolled infrastructure
ThinkAir	Code annotations	Hardware, software and network profilers	Network bandwidth, CPU level of device, RTT	Local historical execution	uncontrolled parallelized methods	Number of servers	Dalvik x86 Multiple servers	Minimal cloud infrastructure, Lacks cluster optimization
COMET	VM-sync at thread level (DSM)	Mobile based T-scheduler	RTT, configurable parameterT	Extended Dalvik	Average speed gain 2.88x, uncontrolled offloading, focuses more on how rather than what to offload	Server is known *a priori*	Dalvik x86 Multiple servers	Uncontrolled infrastructure
Odessa [20]	VM-sync at thread level	Greedy algorithm, lightweight application profiler	Processor frequency, Network history	Runtime environment	Runtime code profiling is preferable, Applications are up to 3x faster	None	Dalvik x86	Minimal cloud infrastructure, Lacks cluster optimization

Table 1: Code offloading approaches from a mobile and cloud perspectives

the cloud analysis. Moreover, this information is periodically updated at runtime for the local variables and asynchronously for the remote variables. Section 6 addresses the concepts of fuzzy logic employed along with the implementation details of the engine.

(2) Mobile virtualization infrastructure
Virtualized instances of the mobile platform are utilized in order to execute the mobile components which are offloaded (at bytecode level) from the device. These instances also implement an automated mechanism that allows to fetch and store code offloading traces into cloud storage. The information stored in the cloud consists of the device information (e.g. bandwidth), mobile application and components information, execution time of the component, type of instance that executed the component, among others. Moreover, instances at the cloud scale up or down based on mobile traffic.

(3) Evidence-based control system
Basically, the control system consists of a cluster which analyzes the code offloading traces by attaching the storage volumes to it. The analysis aims to find offloading patterns, which can be turned into fuzzy rules, data to be fed to the membership functions and information to feed the intervals of the offloading decision variables. The analysis may happen based on different machine learning strategies (e.g. clustering). However, we envision an approach based on neuronal networks due to combining its inference power with the human-like reasoning of fuzzy logic, may allow us to build a system in which the cloud is the expert, and the device asks the cloud for its expertise (*Neuro-Fuzzy*).

(4) Cloud-based messaging
Generally, cloud vendors rely on cloud-based messaging in order to improve the management of real-time data of their cloud-based applications, when those are consumed by mobiles. For example, an e-mail cloud provider may send a notification to trigger a mobile synchronization task (SyncML) that allows the handset to maintain the most recent mailbox information of the user. Asynchronous messaging based on notification services are utilized in order to update the data of the fuzzy logic engine of a specific device.

4. USE CASE: CODE OFFLOADING FOR CLOUD-BASED RESOURCE INTENSIVE MOBILE APPLICATIONS

Several use cases can be envisioned, benefitting from our approach. For instance, a mobile application may improve its execution plan to offload operations in parallel. Similarly, another mobile application may enhance its decision process to make more sophisticated predictions, such as no to offload to a busy server even when suitable connection and low data transfer are met, etc. However, we present in this section, a use case based on resource-intensive task delegation from mobile to cloud. We have studied the delegation of mobile tasks to hybrid clouds in detail and we have developed a Mobile Cloud Middleware [9] framework (MCM) that addresses the issues of interoperability across multiple clouds, transparent delegation and asynchronous execution of mobile tasks that require resource-intensive processing, and dynamic allocation of cloud infrastructure. Moreover, we have developed several successful study cases of data-intensive mobile cloud applications that benefit by going cloud- aware. For instance, Zompopo [21], is an application that consists of the provisioning of context-aware services by processing data collected by the accelerometer of the device, with the purpose of creating an intelligent calendar. CroudSTag [22], is another application that helps with the formation of a social group by recognizing people in a set of pictures stored in the cloud. Similarly, Bakabs [10] is an application, that helps in managing the cloud resources themselves from the mobile, by applying linear programming techniques for determining optimized cluster configurations for the provisioning of Web-based applications. These mobile cloud applications make use of cloud functionality supported by MCM in order to provide richer mobile experience (e.g. video processing) to the end user.

However, the delegation of a mobile task to the middleware happens in a REST-based fashion, using a HTTP mobile client. The byte code of the delegation method can also

be offloaded as a component to the mobile platform (x86) running on the cloud instance. A REST-based operation by nature is dependable on external non-deterministic sources, which are subjected to drastic execution changes. For example, a cloud-based Web application may increase or decrease the number of servers required, as load grows or shrinks, respectively. Consequently, the identification of this candidate mobile operation (involving the REST invocation) and the decision to offload it or not (at thread level [13]), cannot depend exclusively on local logic of the mobile profilers. Moreover, execution information of the service associated with that request may be extracted from previous offloading cases (aka code offloading traces) in order to enhance the future decision process of other devices.

5. CURRENT IMPLEMENTATION

We have developed a partial working prototype of the system, as a proof of concept. The prototype implements the basic functionality needed to show the feasibility of the proposed mobile cloud architecture. At the mobile platform level, we have developed two versions of the fuzzy logic engine (the details of the engine are addressed in the next section), one in Java, which is utilized for simulation purposes of the code offloading model and the other one in C, which is implemented on top of the COMET [13] framework. COMET is utilized, as it provides the basic mechanism to synchronize, at thread level, a mobile application stack with a virtual machine running at the cloud. C-version of the fuzzy logic engine is implemented within the code of the modified dalvik virtual machine provided by COMET. Once, the dalvik virtual machine is recompiled, it is pushed to CyanogenMod7-2 [5], which is an Android [1] based customized operating system, built for Samsung Galaxy S2 i9100 smartphone.

At the cloud level, we have built an Android Open Source Project (AOSP) [1] from source to target a x86 architecture. Android x86 instances are configured to run on Amazon EC2 (Elastic Compute Cloud) infrastructure using small instances (*m1.small* equivalent to 1 EC2 computing unit) with an attached volume of 30 Gb that is used to synchronize AOSP repositories and compile the code locally. One EC2 computational instance is equivalent to a CPU capacity of 2.66 GHz Intel®Xeon™processor. Server was running on 64 bit Linux platforms (Ubuntu). Moreover, an automated Java application that implements jets3t [15] library, is developed and deployed on the server instances, which is utilized to store the code offloading traces in Amazon S3 (Simple Storage Service).

The cluster which is taking care of evidence-based learning is deployed along with MapReduce framework at EC2. However, we are still developing the parallel version of the neuronal network algorithm, which will analyze the code offloading traces. The aim of the algorithm is already described in the earlier section. It mainly pursues to find relations that can be expressed as *if-then-else* constructions that can be pushed as fuzzy rules to the mobile using cloud-based messaging solutions. Currently, this part of the system is being simulated as a black box service.

In the case of cloud-based messaging, we relied on GCM (Google Cloud Messaging) [11] service. GCM is the enhanced notification service provided by Google for sending asynchronous messages to Android devices. Basically, a mobile application that implements the GCM mechanism for receiving messages, first, has to register itself with the GCM server for acquiring a *registration ID*. This ID lasts with the mobile, till it explicitly unregisters itself, or until Google refreshes the GCM servers. Messages are sent to the mobile using this identifier from the application server. An *API key* is required for the application server, in order to pass message data to the GCM service. This key is generated by the developer using the Google APIs console and it is used as the *sender ID* of the message.

An application server sends a message to the mobile by sending the registration ID, the API key and the payload to the GCM servers, where the message is enqueued for delivery (with maximum of 5 attempts) or stored in the case the mobile is offline. GCM allows up to 100 messages stored before discarding the first one. Once, the device is active for receiving and receives the message, the Android system executes an Intent Brodcast for passing the raw data to the specified Android application. GCM do not guarantee the delivery of a message and the sending order. Messages with payload can contain up to 4Kb of data and are encapsulated in a JSON format.

GCM is sometimes unreliable to be used in real-time applications (the details are addressed in evaluation section) as the services are public services and the mobile is competing with huge number of other customers. To address this issue, we have designed a mobile cloud messaging framework based on XMPP (Extensible Messaging and Presence Protocol), which can be integrated with our mobile cloud code-offloading framework. The asynchronous messaging framework is also relevant for our delegation model with our Mobile Cloud Middleware. The description of our mobile cloud messaging framework is beyond the scope of this paper, and is addressed in a different publication [8].

6. A FUZZY LOGIC DECISION MODEL FOR CODE OFFLOADING

The decision of offloading mobile components to cloud may become a variable, which is complex to control in a mobile cloud infrastructure, mainly due to the distributed nature and many real-time constraints of the overall system. However, it is possible to moderate such kind of variables by applying reasoning strategies that enable to infer based on previous experience, in the best case, the most accurate solution and in the worst case, the best estimated result. In this context, approaches such as fuzzy logic are encouraged.

A system based on fuzzy logic has the capabilities to react to continuous changes of the variable to be controlled and by default fosters an inference model based on the comparison of multiple dependent variables. A fuzzy logic system (FLS) consists mainly of four parts (figure 2), *fuzzifier*, *rules*, *reasoning engine* and *deffuzzier*. Basically, a FLS process consists of 1) transformation of crisp sets to fuzzy sets (aka fuzzification). A *crisp set* (CS) is input data which is converted to a *linguistic variable* (LV) that is decomposed into *linguistic terms* (LT), which are assigned to a specific *membership function* (MF); 2) Fuzzy input sets are introduced to the reasoning engine in order to make the inference which is based on a set of rules. Finally 3) the resulting fuzzy output sets are mapped to crisp sets again (aka defuzzification) so that the best result can be extracted.

We have developed our code offloading decision engine based on the previous premises. Basically, the engine con-

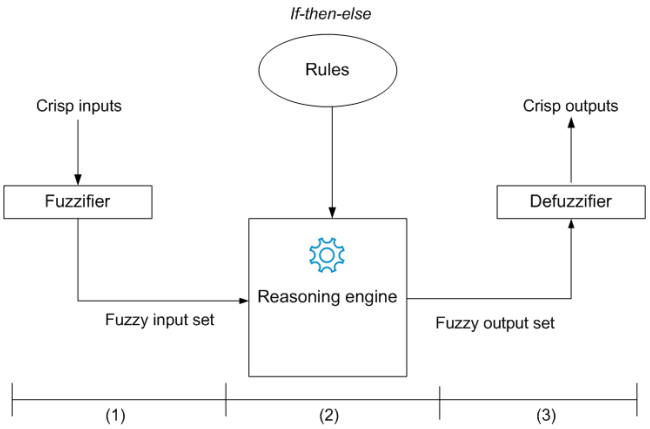

Figure 2: A fuzzy logic system

siders each input parameter (cloud, mobile or any other) as a CS which is transformed to its respective *fuzzy set* (FS). In this case, a LV is created with a general name that identifies the CS. Later, the LV is translated into a possible finite number of FSs, where each FS is associated to a MF, whose values are extracted from the CS and arranged in intervals. For instance, network bandwidth information is refer globally as *bandwidth*, which is divided into intervals *low speed, normal speed and high speed* with values (0, 20), (15, 90), (75, 100) in mbps, respectively. By default, in our model, values follow a trapezoidal distribution function.

Similarly, the variable to be controlled in our code offloading model, the offloading decision, is mapped in the same way as described for the other parameters. However, in this case, the LTs are limited to two options, *local processing* or *remote processing*. Fuzzy rules are created based on basic fuzzy operations, targeting the variable to control. For instance, a simple fuzzy rule to offload to *remote processing* is constructed by combining *high speed* and *instance CPU normal* with an AND operation. Fuzzy sets and rules employed in the analysis are described below. Notice that the code offloading model which is already built, involves just a few elements in order to demonstrate and clarify the potential of the approach. However, a code offloading model based on evidence is expected to be rich in parameters and rules, to provide a sophisticated decision process.

Fuzzy sets considered

Mobile

Bandwidth = *speed_low, speed_normal, speed_high*

Data transfered = *data_small, data_medium, data_big*

Cloud and Code traces

CPU instance = *cpu_low, cpu_normal, cpu_high*

Video execution = *video_low, video_normal, video_high*

Rules considered

remote_processing = *speed_high* AND *data_small*

remote_processing = *speed_high* AND *video_normal*

remote_processing = *data_small* AND *video_high*

remote_processing = *data_medium* AND *video_high*

remote_processing = *data_medium* AND *speed_high*

remote_processing = *cpu_high* AND *video_high*

local_processing = *speed_high* AND *data_big*

local_processing = *speed_low* AND *data_small*

local_processing = *cpu_high* AND *speed_high*

local_processing = *cpu_normal* AND *video_low*

local_processing = *data_medium* AND *video_normal*

local_processing = *video_high* AND *cpu_high*

Furthermore, notice that the relations between the variables depicted as rules in a simple FLS are not clearly visible, and thus we just consider a set of them. However, a more rich set of rules may be found by the cluster that analyzes the code offloading traces at the cloud. Conceptually, in a FLS, the better the rules, the better is the inference.

Since a fuzzy logic system by default infers a decision expressed as the degree of truth to a specific criteria, we tried to quantify how the execution of a mobile component may be segregated to local or remote processing. The grade of truth is a percentage value which represents membership, in other words, it describes how an input, in this case a mobile component, can be classified into a specific group. The classification process consists of estimating the most accurate relation matching based on the fixed set of rules introduced in the system, as explained before. For this analysis, we consider that the mobile component to be offloaded correspond to the source code associated to a resource-intensive REST-based request that follows a mobile cloud delegation model (as described in section 4). Basically, the request consists of uploading a video, which is processed by a MapReduce cluster using a parallel version of a face recognition algorithm in the cloud. Since the cluster may be of different sizes at different points in time, the execution of the video processing request may be higher in some cases and lower in others. The mobile component is analyzed based on described parameters and we assumed that the fuzzy logic engine (rules and membership values) is fed by information extracted from the code offloading traces. Segregation between local and remote processing is also decided in the cloud, based on historical data.

We constructed table 2 in order to simplify the results of the analysis. The table shows how the input parameters are segregated by the fuzzy logic engine to a specific FSs. Based on those FSs, the engine classifies the decision (estimates membership) which can be local or remote. The grade of truth is computed by applying the *center of mass* formula to the decision. Based on the results, we can observe that the decision to offload or not to offload mobile components to cloud is affected by many other aspects of a mobile cloud architecture. In addition, we can clearly observe that even though in some cases the local parameters of the mobile seem to be suitable for offloading, it is not encouraged to offload considering the complete set of variables.

Parameters	Decision	grade of truth
speed_low + data_medium + cpu_high + video_high	local processing	75%
speed_high + data_medium + cpu_high + video_high	remote processing	70%
speed_high + data_medium + cpu_high + video_low	local processing	60%
speed_normal + data_medium + cpu_high + video_high	remote processing	65%
speed_high + data_small + cpu_low + video_high	local processing	78%

Table 2: Offloading decision of the fuzzy logic engine

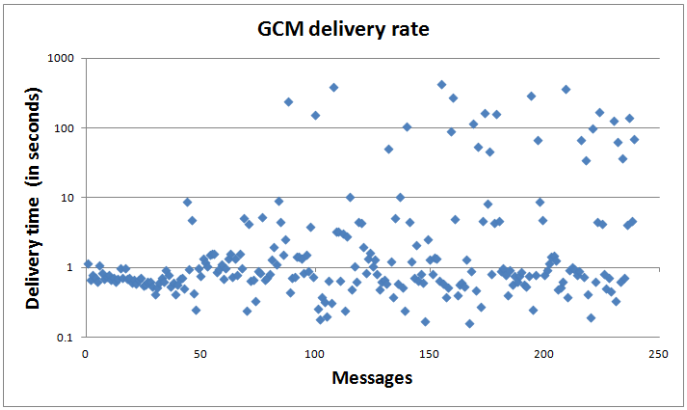

Figure 3: GCM delivery rate

7. EVALUATION OF THE ASYNCHRONOUS MESSAGING MECHANISM

We present in this section, a basic evaluation of the notification mechanism. The aim of the evaluation is to measure the responsiveness of the cloud-based messaging component, as it plays a fundamental role to enhance the offloading decision engine across time. The updates to the fuzzy rules, the decision variables of the cloud are regularly updated at the device, using the notification mechanism.

Several experiments based on the delivery rate of GCM are conducted. The aim of the experiments is to determine the latency between a provider submitting a request and the target device receiving the notification. Mobile device considered in the experiment is a Samsung Galaxy S2 i9100 with Android 2.3.3, 32GB of storage, 1 GB of RAM, support for Wi-Fi 802.11 a/b/g/n and messaging capabilities based on SMS (threaded view), MMS, e-mail, Push mail, IM and RSS. The device is connected via Wi-Fi to a network with an upload rate of $\approx$ 1409 kbps and download rate of $\approx$ 3692 kbps, respectively. Notification server (application server) is configured to run on Amazon EC2 infrastructure using small instances (a small instance has 1.8 GB of memory and up to 10 GB of storage). Server was running on 64 bit Linux platforms (Ubuntu).

GCM application server is written in Java using Servlets technology version 3.0. It is deployed as war project in a Tomcat server. It allows to register multiple devices simultaneously for receiving GCM notifications and to send messages to a specific device. Each GCM message is encapsulated in a JSON payload that includes an index number that identifies the creation of the message and a sending timestamp taken from the server. GCM client application is developed for Android 2.3.3. Once a notification arrives, the payload is extracted from the message and then it is stored in a SQLite database along with receiving client timestamp taken from the mobile.

A message sent to the mobile is fixed to 1024 bytes, which is the maximum payload that can be pushed within the communication channel. Messages are sent 1 per second for 15 seconds in sequence, then with a 30 minute sleep time, later followed by another set of 15 messages, repeating the procedure for 8 hours (240 messages in total). The frequency of the messages is set in this way, in order to mitigate the possibility of being detected as a potential attacker (e.g. Denial of Service) to the cloud vendor and to refresh the notification service from a single requester and possible undelivered data. Moreover, the duration of the experiments guarantee to have an overview of the service under different mobile loads, which may arise during different hours of the day. Experiments were conducted in a Wi-Fi network. Results are shown in figure 3. According to the obtained data, GCM provides longer and shorter delivery times with an average of $\approx$ 380 sec and $\approx$ 0.15 sec, respectively. GCM general delivery time is in average $\approx$ 10 sec. This value is sometimes unacceptable to be used in real-time applications. To address this issue, as already mentioned, we are building our own mobile cloud messaging framework, which is addressed in a different publication [8].

8. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Mobile and cloud computing are converging as the prominent technologies that are leading the change to the post-pc era. Mobile devices are looking towards cloud-aware techniques, driven by their growing interest to provide ubiquitous PC-like functionality to mobile users. These functionalities mainly target at increasing storage and computational capabilities. On one hand, storage limitations of the devices have been overcome by many cloud services provided in the Internet, which are built under different protocols. For example, Amazon S3 and Dropbox [6] provide REST-based access, while Picasa [12] and Funambol [19] provide SyncML-based access. On the other hand, binding computational cloud services such as Amazon EC2 to low-power devices such as smartphones have been proven feasible with latest mobile technologies [4, 3], mostly due to virtualization technologies and their synchronization primitives, enabling transparent migration and execution of intermediate code. Moreover, multiple research works have proposed different code offloading strategies to empower the mobile application with cloud resources. Most of the solutions try to overcome the problem by granting the mobile, a local context logic, which is used to decide whether a mobile component is offloaded or not to cloud.

However, many other aspects of the mobile cloud architecture may affect the offloading decision process of a device. We proposed in this paper a code offloading approach from a different perspective. Basically, our hypothesis is that code offloading may fail in some cases, as the information currently being utilized to make the decision is not enough to measure a real benefit for the device. We have implemented

a partial prototype of our system based on principles of fuzzy logic. According to the results, it is possible to enrich the offloading decision process asynchronously by analyzing code offloading traces and introducing cloud infrastructural parameters. The contributions of this work are envisioned as an approach that enables to enhance the code offloading decision process of a mobile device with cloud power, "Evidence-based mobile code offloading" approach.

As future research directions in this domain, we are interested in fully implementing the current simulated component of our mobile cloud architecture, and in demonstrating the approach with possible case studies. Basically, we want to develop a parallel version of the neural network algorithm that can be deployed on a MapReduce cluster so that it can be utilized to analyze big repositories of code offloading traces. Moreover, we are also interested to explore and compare other machine learning strategies (e.g. clustering) that can be utilized in a parallel fashion in order to find *if-then-else* patterns that can enrich our fuzzy logic engine.

Furthermore, while it has been shown that the offloading decision process may be enriched from a global infrastructure perspective in this paper, we are interested to compare our fuzzy logic mechanism with other offloading decision engines, in order to show the energetic benefits of splitting the the decision process between the mobile and cloud.

9. ACKNOWLEDGMENTS

This research is supported by the Tiger University Program of the Estonian Information Technology Foundation, the European Regional Development Fund through the EXCS, Estonian Science Foundation grant ETF9287, Target Funding theme SF0180008s12 and European Social Fund for Doctoral Studies and Internationalisation Programme DoRa.

10. REFERENCES

[1] Android Open Source Project. Welcome to Android. http://source.android.com/.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech.*, 2009.

[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 sixth conference on Computer systems*, pages 301–314, 2011.

[4] 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*, pages 49–62. ACM, 2010.

[5] CyanogenMod. A customized aftermarket firmware distribution. http://www.cyanogenmod.org/.

[6] Dropbox, Inc. Dropbox - Simplify your life. https://www.dropbox.com/.

[7] S. Durga and S. Mohan. Mobile cloud media computing applications: A survey. In *Proceedings of the Fourth International Conference on Signal and Image Processing 2012 (ICSIP 2012)*, pages 619–628. Springer, 2013.

[8] H. Flores and S. N. Srirama. Mobile cloud messaging supported by xmpp primitives. In *Proceedings of the Fourth ACM Workshop on Mobile Cloud Computing and Services (MCS 2013)*. ACM, 2013.

[9] H. Flores, S. N. Srirama, and C. Paniagua. A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, pages 87–94. ACM, 2011.

[10] H. Flores, S. N. Srirama, and C. Paniagua. Towards mobile cloud applications: Offloading resource-intensive tasks to hybrid clouds. *International Journal of Pervasive Computing and Communications*, 8(4):344–367, 2012.

[11] Google, Inc. GCM - Google Cloud Messaging for Android. http://developer.android.com/guide/google/gcm/.

[12] Google Inc. Picasa - web albums. https://picasaweb.google.com/home.

[13] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. Comet: code offload by migrating execution transparently. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, pages 93–106. USENIX, 2012.

[14] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, 3(3):66–73, 2004.

[15] jets3t. jetS3t - An open source Java toolkit for Amazon S3 and CloudFront. http://jets3t.s3.amazonaws.com/toolkit/guide.html.

[16] G. J. Klir and B. Yuan. *Fuzzy sets and fuzzy logic*. Prentice Hall New Jersey, 1995.

[17] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[18] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[19] A. Onetti and F. Capobianco. Open source and business model innovation. the funambol case. In *Int. Conf. On OS Systems Genova, 11th-15th July*, pages 224–227, 2005.

[20] 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*, pages 43–56. ACM, 2011.

[21] S. N. Srirama, H. Flores, and C. Paniagua. Zompopo: Mobile Calendar Prediction based on Human Activities Recognition using the Accelerometer and Cloud Services. In *5th Int. Conf. On Next Generation Mobile Applications, Services and Technologies (NGMAST)*, pages 63–69. IEEE CS, 2011.

[22] S. N. Srirama, C. Paniagua, and H. Flores. Social group formation with mobile cloud services. *Service Oriented Computing and Applications*, 6(4):351–362, 2012.