View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

Improving the reliability of an offloading engine for Android mobile devices and testing its performance with interactive applications

Martí Griera
Berlin, 9th October 2013

Master's Thesis

Supervisor: Prof. Dr. Katinka Wolter
Assisting Supervisor: Prof. Dr. Mesut Günes

Freie Universität Berlin
Department of Mathematics and Computer Science
Institute of Computer Science
Working group of Dependable Systems

Statutory Declaration

I hereby declare to have written this thesis on my own. I have used no other literature and resources than the ones referenced. All text passages that are literal or logical copies from other publications have been marked accordingly. All figures and pictures have been created by me or their sources are referenced accordingly. This thesis has not been submitted in the same or a similar version to any other examination board.

Berlin, 9th October 2013

Martí Griera Jorba

Acknowledgements

I especially would like to thank my collaborator Joan Martínez and my supervisor Katinka Wolter. In addition, I can not express my gratitude to Milena Lütz for her unconditional support. Many thanks too to Victòria Jorba, Miquel Jorba, Dagmar Semder and Fabian Lütz for their support from the distance, and to Juliane Schicketanz and Meritxell Gimenez for their encouragement from the proximity.

Martí Griera Jorba

Nothing worth having comes easy...

Abstract

The already ubiquitous though still growing market of mobile devices disposes of an increasingly prolific offer of software applications, which makes them even richer. However, the inherent resource constraints of these devices, such as processing, memory, storage or battery capacity, are limiting the performance of the more resource-hungry applications. Taking advantage of the relatively strong network connection capabilities of the mobile devices, many approaches have emerged in the last years proposing mobile cloud computing as a solution. Of these, mobile computation offloading plays an important role. It is especially useful for intensive processing applications, as it consists of sending a part of the computation load of a mobile device to be processed in outside surrogates such as the cloud infrastructure. Thanks to this technique, the performance of the applications can be notably enhanced, while reducing the energy consumption of the devices.

A study of the current computation offloading scene is conducted in this paper, analyzing many of the systems developed recently. The trend of these systems is to decide dynamically -on runtime- whether it is worth or not to offload a task. In order to take the right decision, many conditioning aspects and parameters can be considered.

This thesis offers a dynamic decision offloading approach that focuses on improving the applications' performance. The system will consider that it is worth to offload a task when its estimated execution time on the mobile device is greater than the sum of its estimated execution time on the surrogate plus the predicted costs of the data transfers.

Mobile computation offloading is commonly used in areas like multimedia processing, vision, recognition, graphics, gaming or text processing. Concrete examples are applications such as face detection, speech recognition or the artificial intelligence of a game. Observing that the heavy computation tasks of these applications have significantly variable execution times depending on its input (how big is the image where faces must be detected, how long is the audio file where the speech must be recognized, which is the difficulty level of the artificial intelligence, etc.), a system to estimate the execution time of a task depending on which are its input parameters is designed. This system computes the estimations statistically from past observations, and is based upon a nonparametric regression technique.

In order to evaluate the presented offloading system, an implementation of it is carried out (extending a simple offloading engine for Android) and many tests are run, checking the behavior of the system with some interactive applications, such as a chess game. The results obtained from the experimentation indicate that most of the taken offloading decisions are correct and it is verified that the overhead produced by the decision making is small enough to affect only minimally the overall performance.

As a conclusion, the mobile computation offloading approach proposed in this thesis is valid to improve the performance of many applications, but further work must be done in order to increase its ease of use and compatibility.

Table of Contents

1 Introduction....1

2 Theoretical Background....6

2.1 Taxonomy of the main aspects of an MCO system....6

2.1.1 Motivation....7

2.1.2 Partitioning....7

2.1.3 Decision-making....9

2.1.3.1 Parameters....12

2.1.4 Offloadable entities....14

2.1.5 Serialization....15

2.1.6 Virtualization....15

2.1.7 Networking, mobility and fault tolerance....16

2.1.8 Infrastructures....17

2.1.9 Security, privacy and trust....18

2.1.10 Applicability....19

2.2 Related works....20

3 The proposed system....24

3.1 Theoretical approach....24

3.1.1 General description....24

3.1.2 Architecture....26

3.1.3 Decision-making....28

3.1.3.1 Parameters....30

3.1.4 Serialization, virtualization and fault tolerance....34

3.2 Automated estimation system of task execution times....34

3.2.1 Overview....34

3.2.2 Database design....36
3.2.3 The nonparametric regression technique....36
3.3 Implementation....38
3.4 Evaluation....40
3.4.1 Experiments' setup....40
3.4.2 Results....41
3.4.3 Interpretation....46
3.5 Further work....47
4 Conclusions....49

List of abbreviations

2G	Second Generation
3G	Third Generation
4G	Fourth Generation
A-GPS	Assisted Global Positioning System
AESTET	Automated Estimation System of Task Execution Times
AI	Artificial Intelligence
AIDL	Android Interface Definition Language
API	Application Programming Interface
ARM	Advanced RISC (Reduced Instruction Set Computer) Machine
B	Byte(s)
CBIR	Content-based image retrieval
CC	Cloud Computing
CLR	Common Language Runtime
CO	Computation Offloading
CORBA	Common Object Request Broker Architecture
GHz	Gigahertz(es)
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
iOS	i (from Apple products) Operating System
ISA	Instruction Set Architecture
JAR	Java Archive
JNI	Java Native Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KB	Kilobyte(s)
Kb/s	Kilobit(s) per second
k-NN	k-Nearest Neighbor(s)
KVM	Kernel-based Virtual Machine
LTE	Long Term Evolution
MB	Megabyte(s)
Mb/s	Megabit(s) per second
MCC	Mobile Cloud Computing
MCO	Mobile Computation Offloading

MDO	Mobile Data Offloading
MHz	Megahertz(es)
OS	Operating System
OSGi	Open Services Gateway initiative
PaaS	Platform as a Service
PANDA	Policy-based Model-driven Pervasive Service Creation and Adaptation
PC	Personal Computer
PDA	Personal Digital Assistant
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTT	Round-Trip Time
SaaS	Software as a Service
SSL	Secure Sockets Layer
UI	User Interface
UMTS	Universal Mobile Telecommunications System
VM	Virtual Machine
WAP	Wireless Access Point
WAR	Web application ARchive
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WMN	Wireless Mobile Network
WUI	Web-based User Interface
XML	Extensible Markup Language

1 Introduction

Advancements in computer technology have expanded the presence of computers and network access -the combination of which will be of special interest in this thesis- in a wide variety of mobile devices, from laptops, PDAs, tablets or mobile phones to A-GPS devices, sensors or autonomous robots (furthermore, according to the concept of the Internet of Things [IoT13], this presence will keep expanding). Of these devices, the mobile phones are the main computing wave, especially the complex smartphones, gaining everyday more market and popularity over desktop computers and laptops. In 2012, the smartphone shipments already exceeded the shipments of normal phones [Jun12]. These mobile devices have many resource constraints in comparison to a desktop computer, such as processing, memory, storage or battery capacity. In addition, the users want them to be smaller and thinner everyday and, at the same time, to be more powerful, two features that play against each other.

Responding to this demand, the smartphones' resources have increased considerably, considering that the speed of the top model smartphone processors incremented during the last decade from less than 200 MHz to more than 1GHz [Kem10]. However, when a processor's clock speed doubles, the power consumption of the device nearly octuples [Kum10], and the battery capacity cannot follow a growth rate big enough to provide this much power. This is identified as one of the main bottlenecks on the mobile devices resources.

Furthermore, the applications for smartphones have also proliferated a lot recently. The availability of applications increased largely thanks to the rise of application stores, through which the process of finding and installing applications has become much simpler for the end users. Another reason of the large offer of applications is that now, not only software companies develop them, but also small developers and hobbyists (the applications for the Apple App Store have incremented from 500 to more than 200,000 within two years [Kem10]). Many of these applications put pressure on the manufacturers to keep expanding the capabilities of the mobile devices.

On the other hand, the modern mobile devices support several types of network interfaces, e.g. Ethernet, Bluetooth, Wi-Fi, GPRS or even WiMAX, but the latency with the corresponding networks is not negligible and the bandwidth not always as wide as it should. In each situation the most appropriate interface

should be selected according to their availability, but no network provider can support such switches [CDA11]. However, this is partially compensated with the fast networking technology evolution.

The cellular networking technology -the networking through cell towers-, has grown rapidly, from the 14.4 Kb/s allowed by the GSM networks (known as 2G), evolving to the UMTS networks (3G) and now coming to the LTE networks (4G), that are expected to provide around 100 Mb/s [Kem10]. Moreover, the novel Femtocell devices will help relieving the cellular networks' shortcomings [Din11]. Simultaneously, the local area wireless networks (WLAN), such as Wi-Fi, have increased in bandwidth, too. The WLANs are becoming more and more used by the smartphones' users; the Wi-Fi networks already drive about 65% of the total mobile data traffic [Lee13]. As exchanging data with a nearby wireless access point -of a Wi-Fi network- requires less energy than exchanging data with a potentially far cell tower, the increasing use of the Wi-Fi networks already save around 55% of battery power, but this is still not enough to solve the energy problem on the mobile devices.

In this context, with a growing market of inherently resource-constrained mobile devices that dispose of a gigantic offer of software applications, the concept of mobile cloud computing (MCC) appears -enabled by a relatively strong network technology- as an alternative to help reducing the increasing gap between the actual resources of the devices and the demanded ones by many resource-hungry applications.

MCC consists of enhancing the capabilities of mobile devices with the cloud computing infrastructure, especially with the principle of delivering applications and services for the mobile devices. In the last years, MCC has become very successful, and market research predicts that by the end of 2014 mobile cloud applications will deliver annual revenues of 20 billion dollars [Chu11]. Two main groups of MCC can be identified, mobile data offloading (MDO) and mobile computation offloading (MCO).

Given the reduced storage capacity of the mobile devices, the idea of MDO is to upload to the cloud space-consuming data (typically multimedia data) or backups of the device's data. When the data needs to be operated, it can be downloaded again or managed through browser-based applications residing wholly in the cloud. There exist many applications separated into a light weight client and a heavy weight server hosted in the cloud, for example the music search service Shazam [Sha13] or the image search service Goggles [Gog13].

On the other hand, MCO takes advantage of the richer computational resources of the cloud infrastructure to process part of the computation load of a mobile device there. A simple Google search or the use of the above mentioned browser-based applications can be considered MCO, as the mobile device indeed does less processing thanks to this services, but actually the mobile device could not perform this activities unless a huge amount of data would be downloaded. In this thesis, MCO is understood as having both the possibility of executing a part of the computation -the intense part- of a native mobile application in the mobile device or offloading it to a remote cloud infrastructure (or nearby idle computers). The remote servers will be called surrogates.

The feasibility and utility of MCO have already been proved [Kum12]; thanks to MCO the performance (hereafter, performance refers to the execution time performance) of the applications can be notably enhanced, while reducing the energy consumption of the devices. Therefore, MCO can help softening the above mentioned bottleneck on the power supply for the increasingly demanding processors. Energy is a primary constraint for mobile systems; however, this thesis will focus on improving the performance of the applications rather than reducing the energy consumption, since in most cases one thing entails the other one. Nevertheless, it will be of interest to identify under which circumstances both things are not equivalent.

Many offloading systems and frameworks appeared in the last years, but the majority of application developers still have a lack of awareness of the advantages that MCO provides. Moreover, the ease of use of these systems is still something to be improved, and the programmers think it is not worth the burden that the use of MCO supposes.

However, the scope in which MCO is more advantageous is well-known, and it comprises intensive computation applications in areas like multimedia processing, vision, recognition, graphics, gaming or text processing [Kum12]. Concrete examples are applications such as face detection, speech recognition or the artificial intelligence of a strategy game. Observe that the heavy computation tasks of these applications have significantly variable execution times depending on its input (how big is the image where faces must be detected, how long is the audio file where the speech must be recognized, how many enemies controls the artificial intelligence and how smart they are, etc.).

Thus, the main objective of this thesis is to conceive a system designed to improve

the applications' performance through MCO, and evaluate if it can be useful with real life applications. At the same time, this thesis aims to provide a simple and easily extensible offloading system usable for any kind of future experimental purposes, hence an implicit goal will be to keep its modularity, adaptability and availability [FUB13]. Research will be done about the main conditioning aspects and parameters characteristic of an offloading system, and many systems developed recently will be reviewed.

The system presented here will not partition applications identifying the potentially offloadable parts; this will have to be done by the applications' programmers at development time, and many other aspects out of the focus of this thesis will be simplified, too. Rather, the fundamental goal of the system will be to be able to decide at runtime whether a potentially offloadable task will be executed locally or remotely. Offloading will be considered worth when the estimated execution time of the task on the mobile device is greater than the sum of its estimated execution time on the surrogate plus the predicted costs of transferring the data.

In order to solve the formulated inequality, it will be necessary to predict both the execution time of a task in the mobile device and the surrogate, and to predict the costs of sending the request and receiving the answer over the network. While the network costs will be roughly estimated, a derived objective of this thesis will be to accurately estimate the execution times of the tasks, and an automated system will be designed for that matter. The main point of the system will be to forecast the execution time of a task depending on which are its input parameters (a significant property, as described above). This system will compute the estimations statistically from past observations, and will be based upon a nonparametric regression technique. This type of system is usually used to predict the execution time of large tasks in distributed systems or grid computing [Wol08], thus the overhead of calculating the estimated execution time is not relevant. However, it will be a challenge to see if the overhead can be reduced enough to fit in the decision-making process of MCO.

To verify the proposed approach, a practical implementation will be built almost from scratch, as a motivation of this thesis is to deal closely with the mobile computation offloading. A simple offloading engine for Android [GM13] will be extended, mainly improving its reliability through better forecasting capabilities. The automated estimation system of task execution times (hereafter: AESTET) will be added and the network data transferring time prediction mechanism will be improved, reimplementing much of the original engine. The resulting

implementation will not be expected to be as competent as the current MCO systems, but it will be enough for the testing purposes of this work. Its effectiveness and efficiency will be evaluated running it on some interactive applications, such as a real chess game or a testing application prepared for this matter.

Summarizing the above exposed, the research in this thesis will aim to give answers to the following questions regarding the MCO system that will be presented:

- Can it improve the performance of mobile devices' applications?
  - Does it decide correctly when is it worth to offload a task?
    - Are the rough network data transferring time forecasts enough for the approached system?
    - Is AESTET accurate enough?
  - Does AESTET produce too much overhead?
  - Is it helpful with real life applications?
- Can it be affirmed that by improving the performance the energy consumption is reduced, too?
- Is it usable for future experimentation?

In the following chapter, a study of the current computation offloading scene is conducted, reviewing many of the systems developed recently and aiming to identify the main conditioning aspects and parameters characteristic of an offloading system. In chapter 3, the MCO system proposed in this thesis will be presented. First, in section 3.1, it will be decided which aspects the system is going to cover (besides the ones required for the main purposes of the system) and which are going to be simplified or ignored. In the same section, the system architecture, key components and design of the system will be explained, followed by the also theoretical description of AESTET in section 3.2. The next section will show the most relevant implementation details of the prototype of the system, which will be used for the evaluation of the work in the next section. Section 3.4 will include the results of the experiments and the difficulties found will be discussed. The future outlook will be commented in the last section of this chapter. Finally, chapter 4 will synthesize the most important points of the thesis in the conclusions.

2 Theoretical Background

2.1 Taxonomy of the main aspects of an MCO system

The term cloud computing (CC), or colloquially, the cloud, does not have a commonly accepted unequivocal scientific or technical definition, but it is used to describe diverse computing concepts involving a large number of computers connected through a network. Its success and strong future outlook have already been widely recognized. The concept of CC usually involves having a data center, a hardware facility with many servers typically built in a low populated place, with high-speed networks and a high power supply stability. This offers many advantages, allowing to provide the users with different cloud services, mainly:

- Infrastructure as a service (IaaS): is the lowest model, provides the most basic service allowing the users to use the infrastructure directly, e.g. servers, virtual machines, load balancers, networks, storages, etc.

- Platform as a service (PaaS): abstracts from the IaaS model and offers a platform to work on, e.g. middleware services, operating systems, databases, web servers, development tools, etc.

- Software as a service (SaaS): abstracts from the PaaS model and provides concrete software services, e.g. application programs, email, games, etc.

These services are provided at low cost by cloud providers, e.g. Google, Amazon or Salesforce [Din11]. The users can elastically utilize cloud resources in an on-demand fashion, thus making CC very suitable for rapidly provisioning and releasing the mobile devices' applications. With the boom of mobile devices and applications, the term MCC is introduced.

As outlined in the first chapter, MCC can be grouped into MCO and MDO. Thanks to MCO, the battery lifetime of a mobile device can be extended and its processing power improved. MDO extends the storage capacity of a mobile device and also improves the reliability, as having the data backed up in the cloud is a guarantee for most users.

The concept of computation offloading (CO) already existed before MCO, but it was focused on offloading the computation in static environments, i.e. a server connected to a stable network . In this case, the tasks to be offloaded are usually

huge and are sent to various computers, e.g. distributed systems or grid computing.

The following subsections will point out the main aspects affecting the MCO systems, mentioning many properties actually inherited from CC or CO.

2.1.1 Motivation

First, an MCO system can be oriented towards different goals:

- Improving performance
- Saving energy
- Reliability
- Context awareness

The main differentiation here is whether the system will focus on saving energy or improving performance, or both. As will be explained later, if a system focuses on improving one of this two goals, the other will be usually achieved as a collateral effect too, but there are some subtleties about this relation.

As described above, improving the reliability is a property that fits better to MDO, but it is also important in the context of MCO. Many MCO approaches propose executing in the surrogates not exactly the same that would have been executed in the mobile device, but rather having two versions of the potentially offloadable tasks, one for the mobile device and one for the surrogate. Thus, the task executed in the surrogate can be more complex and precise than the one in the client, e.g. a face detection application can have more exhaustive algorithms in the server side than in the client, producing a better recognition when the detection task is executed in the surrogate.

More recently, it has appeared the trend in the MCO systems to aim at context awareness [CDA11, Kum12, Din11]. This refers to perceive the user's state and surroundings (e.g. user's location, preferences, data types, network status, device environments, etc.) in order to infer context information from it, with adaptive mechanisms based on this information that are able to provide the appropriate services to each user and situation.

2.1.2 Partitioning

An important part of the job of an MCO system is to divide the applications into

the parts that must be run on the mobile device and the parts that can be potentially offloaded. The handling of the peripherals -e.g. the camera- and the I/O interactions with the sensors of the device should not normally be offloaded, as well as the communication with the user interface (UI). As will be seen in 2.1.7, if an offloading action fails due to network problems, a typical solution is to re-execute the task locally. Considering this, it is also a good practice that the potentially offloadable parts do not communicate with external elements or remote machines. For example, a task containing an e-commerce transaction could be offloaded and the transaction could be effectively executed in the surrogate, but then the connection could be lost and the task re-executed again locally, thus doing two times the same transaction. However, these omissions are not categorical and there are studies that address how to circumvent them [Sti10].

There are two possibilities on how to partition an application and identify the tasks suitable for offloading:

- Human-made: the MCO system does not partition the system, instead, the programmer of the application must separate during the development the not offloadable parts from the parts that may be offloaded. This requires an extra burden on the programmer and more involvement, but in contrast the partitions can be more optimal and the overall efficiency should be better (more energy savings and performance).

- Automated: the MCO system auto-determines the partitioning scheme of the application automatically. It is desirable to perform it in development time too, to avoid the very high overhead for analyzing the program that it would produce during execution. This complex approach has been deeply studied [Til06, ZLi01]; it might require techniques such as static analysis and dynamic profiling [Chu11]. In scenarios where the application information is unknown [Xia07], no partitioning may be done and the entire program is either offloaded or executed locally. An MCO with automated partitioning might be used more easily by the applications' developers, but at the same time is likely to be less efficient.

Another important aspect of the partitioning process is the partitioning granularity. A fine-grained partition would identify as offloadable only the truly intensive computing parts of an application, while a coarse-grained partition would not isolate that precisely. Depending on the partition granularity, diverse offloadable entities may result. These will be explained in detail in subsection 2.1.4.

2.1.3 Decision-making

Deciding whether offloading a task is worth to be done or not is one of the main challenges of MCO. Generally speaking, offloading is beneficial, whenever the gained efficiency outweighs the costs involved [CDA11]. The offloading decisions can be classified as:

- Static: the offloading decisions are made beforehand, not during the application's runtime. The properties of the potentially offloadable tasks of the application are analyzed in order to decide the more convenient action, so the partitioning must have been done previously, in development time. This approach is valid only when the parameters needed to evaluate the offloading condition (explained in 2.1.3.1) can be accurately predicted in advance, and has the advantage of low overhead during execution. Static decisions can be human-made or automated.

- Dynamic: the offloading decisions are made at runtime by the MCO system, just before starting to execute any potentially offloadable task, being able to adapt to many dynamic conditions such as the changing connection status or the fluctuating bandwidth. Dynamic decisions need to predict even more variable parameters than the static ones, and this is done at runtime, so the efficiency of the prediction mechanisms will be essential in order to avoid producing too much overhead.

In the recent years, there are fewer papers suggesting MCO with static decisions, and currently the majority are based on dynamic decisions [Kum12]. However, selecting the appropriate decision-making type is heavily dependent on the nature of the offloadable functionalities on which the MCO system focuses (explained in subsection 2.1.10).

Much of the literature about MCO studies under which circumstances offloading computation from a mobile device is beneficial, by means of performance or energy saving. While the general equations that describe these two criteria can be easily stated (*Eq. 1* and *Eq. 4*), evaluate them turns to be the real challenge of the MCO systems, as the variables represented in the equations cannot be effortlessly obtained due to the dynamic nature of the mobile devices' environment. Much of the research focuses on prediction mechanisms for these parameters.

For performance improving, the offloading offloading condition can be simply

formalized in the following way:

$$\frac{I}{S} + \frac{D}{B} < \frac{I}{M} \tag{1}$$

where I is the number of instructions of the potentially offloadable task, S is the computing speed of the surrogate (instructions/second), D is the size of the data to be transferred (bytes), B is the bandwidth of the network (bytes/second) and M is the computing speed of the mobile device (instructions/second). Note that the left side of the inequality corresponds to the offloading time (cost of executing the task remotely plus cost of transferring the data) and the right side to the cost of executing the task locally. This way it becomes evident, that the execution time saving criterion indicates to do offloading when this inequality evaluates to true. The surrogate's computing speed may vary but, in general, it can be assumed to be tens of times superior (F times, below) to the computing speed of the mobile device (some cloud vendors can guarantee a minimum level of performance). Considering this, and that transferring the data actually means sending a request and receiving a response, the inequality in *Eq. 1* can be rewritten as:

$$\frac{I}{F \cdot M} + \frac{D_s}{B_u} + \frac{D_r}{B_d} < \frac{I}{M} \tag{2}$$

Here D_s stands for the size of the data to be sent and D_r for the size of the data to be received. The bandwidth is also split in the respective cases: B_u refers to the upload bandwidth and B_d to the download one.

Observe that according to *Eq. 2*, no time would be needed to send a very small request or receive a very small response. However, there is a propagation time between the moment that the request leaves the mobile device and the moment when it reaches the surrogate, and vice versa. The sum of these two times is called the round-trip time (RTT). In addition to the RTT, transferring data through the network also requires an initial setup time that is often ignored [Kum12]. Let R be a single variable representing the sum of these two small times, then:

$$\frac{I}{F \cdot M} + R + \frac{D_s}{B_u} + \frac{D_r}{B_d} < \frac{I}{M} \tag{3}$$

which is the expanded equation for the performance criterion.

On the other hand, when the aim is energy saving, the offloading condition can be represented as follows:

$$P_i \cdot \frac{I}{S} + P_t \cdot \frac{D}{B} < P_c \cdot \frac{I}{M} \quad (4)$$

This equation is identical as *Eq. 1*, except for being each addend multiplied by its corresponding power consumption. Thus, the generic energy saving criterion derives from the performance criterion. Each of the addends (I/S, D/B and I/M) is expressed in seconds, which multiplied by watts (joules/second) give joules as a result, the energy unit. P_i corresponds to the power consumed by the mobile device's processor while being idle, as it is assumed to be idle while waiting for the surrogate's response[1] and P_c is the power consumed when the processor is computing[2]. It can be said, without loss of generality, that for all the mobile devices $P_i < P_c$ is true. The power consumed when transferring data, P_t, totally depends on the type of network, e.g. Wi-Fi consumes considerably lesser power than 3G.

Now, breaking down the times like in *Eq. 3*, the inequality in *Eq. 4* can be rewritten as:

$$P_i \cdot (\frac{I}{F \cdot M} + R) + P_t \cdot (\frac{D_s}{B_u} + \frac{D_r}{B_d}) < P_c \cdot \frac{I}{M} \quad (5)$$

Observe that the power consumed by the processor while being idle, P_i, is associated with the time R for simplicity. It is not exact though, as R is the sum of the RTT and the network initial setup time. Similarly, there is a transferring power P_t associated with both the data sending and receiving times, although there might be small differences between the power for sending data and the power for receiving data.

It can be deduced from the equations (*Eq. 1* already shows it) that, even if the surrogate's computation capabilities were infinitely faster ($F=\infty$) than the mobile device's ones, if $D/B > I/M$ is true, then no offloading should be done. As a conclusion, both criteria -performance improving and energy saving- are, above all, dependent on the data D to be transferred, the bandwidth B and the amount of computation I to be done. For both criteria, applications with light communication (small D) and heavy computation (large I) are appropriate. Figure 1 illustrates the relations between D, I and B:

---

1 There are approaches that aim to use the processor during that time.

2 Many mobile devices have the ability to run their processors at different frequencies depending on the computation load of each moment, in order to save energy [CSR13]. In this thesis it will be assumed that P_c is the power consumed by the processor at its maximum frequency (as it should be when running supposedly intensive processing tasks).

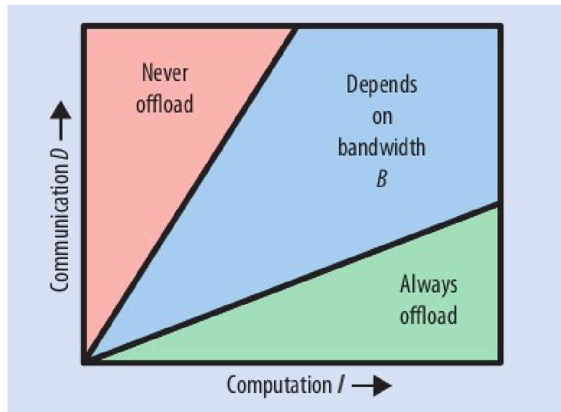

Figure 1: Relations between the size of the transferred data (D), the bandwidth (B) and the amount of computation (I) in the context of MCO [Kum10].

Note that the previous equations present an important assumption: the number of instructions of the task is the same whether it is executed locally or in the cloud. While the concept of computation offloading primarily means to execute the same task in another machine in order to reduce the computation load, in MCO it is a common approach to have a variation of the task in the surrogate, typically running heavier and more accurate algorithms in the surrogate to take advantage of its richer resources. However, this is done when the computation speed of the server is big enough to compensate it, so the formalization model of the presented equation is still generic.

2.1.3.1 Parameters

In order to decide whether to start an offloading process or not, an MCO system must evaluate either *Eq. 3* (for performance) or *Eq. 5* (for energy saving), or both. A variety of prediction mechanisms are proposed to forecast the many parameters in the equations. If the MCO system uses static decisions, this can be partly done by humans, but prediction tools are still needed. In the case of a dynamic decision MCO system, the decision must be made at runtime, just before the execution of a potentially offloadable task, thus the forecasting mechanisms will have to be especially efficient.

The parameters affecting an offloading decision can be of different types: static information, hardware, network or other contextual information. The following list gathers the parameters seen in *Eqs. 1* to *5*, describing how to forecast their values:

- P_i, P_c, P_t : the power consumed when idle, computing or transferring data varies from one mobile device to another, thus an MCO system only needs to calculate them once[3]. There are different P_t for the different network interfaces: P_{3G}, $P_{Wi\text{-}Fi}$, etc., so the right one can be chosen only dynamically, depending on the network used at that moment.

- *I*: the amount of instructions that a task requires could always be similar, so no forecasting would be needed. Otherwise, there are many techniques to estimate the amount of the computation of a task [Wol08], such as static code analyzing [Rei94] and statistical estimation [Ive96]. These techniques usually use the input parameters of a task (or the size of the input) to do the forecasts. Sometimes the input is known in advance, otherwise the predictions will have to be done just before the execution of the task. This is especially challenging in the context of MCO, because a big overhead would be prohibitive.

- D_s, D_r : the amount of data to be transferred depends on what is actually going to be transferred to the surrogate, and this will be seen in detail in subsection 2.1.4. Often, it can be assumed that it will not be necessary to send the task itself, as the surrogates will already have a copy of it. In any case, the input parameters of the task and the results of the execution must be sent and retrieved, respectively. The size of this input is very variable, while a chess game requires only a few bytes, an image processing application might require a whole picture file. However, in applications dealing with multimedia files, it is common to work with data already offloaded to the surrogates (thanks to MDO), so only pointers to this data are needed[4]. In either way, no prediction is needed for D_s as it can be obtained directly before the execution of a task. The prediction of D_r varies radically depending on each task.

- *S*, *M*: the computation speed of the surrogate *S* is a known value, usually considered constant. On the other hand, the computation speed of the mobile device *M* depends on the device, thus the MCO system only needs to calculate it once. It can be deduced directly from the hardware

---

3 It is not easy to obtain these values though, as the mobile devices' OSs do not provide functions in their APIs to do so. This happens because this information is not directly readable from the mobile devices' hardware, and there are no sensors accurate enough to calculate them.

4 Nonetheless, applications dealing with real-time data (e.g. a face recognition application that uses as input a newly captured picture) have no other option than sending the whole input to the surrogate.

properties or measured through small tests.

- B_u, B_d : the upload and download bandwidths can only be forecast through experiencing, as the network state is external. A possible approach is to monitor the bandwidth and predict it with a Bayesian scheme [Wol08].

- R: as well as the bandwidth, only probabilistic schemes can predict the round-trip time.

In general, Wi-Fi networks offer more stable bandwidth and RTT than cellular networks (like 2G or 3G).

2.1.4 Offloadable entities

This subsection gives an answer to the question of which data is going to be sent to the surrogate. It is related with the properties of the potentially offloadable tasks identified when partitioning the application.

An offloadable entity is the input information needed to do the offloaded computation in the surrogate. Depending on the type of offloadable entities that an MCO system uses, some traits about the general behavior of the MCO system can be extrapolated. Thus, many classifications [Kum12, Jun12, CDA11] extend the meaning of the term offloadable entity, but in this thesis it will be limited to the defined. The most remarkable types of offloadable entities are the following, listed from less to more amount of data:

- Feature: only the dataset strictly needed to solve the computational problem in question.

- Method: method calls with the needed data.

- Image: an image of the program code or low-level code selected by the OS scheduler (the surrogate maintains a state corresponding to the one of the mobile device's process [Chu11]).

In the case of features, the application is partitioned typically by the application's programmer at development time, as the level of optimization required is complex for an automated mechanism. The same happens with the identification of potentially offloadable methods [Cue10].

With methods, the execution of subroutines is transferred to the cloud. The ability of most of the programming languages to offer techniques such as reflection, introspection and method wrapping is exploited. As the MCO system will be working at the object and class levels [Yan08], it can take advantage of the many distributed object frameworks and technologies, such as Java RMI, CORBA, OSGi, .NET Remoting or RPC.

Working at the level of features involves more burden on the programmer, whereas methods and images reduce the effects of offloading in application development.

2.1.5 Serialization

Once having defined the data to be sent to the server, the next question to answer is how to send it. Serialization is the technique used to translate data structures or object states into a format that can be stored and retrieved again later. An MCO system can use a wide variety of already existing serialization formats, such JSON, XML or MessagePack [Msg13]. Depending on the implementation of the MCO system, the programming language might offer support to serialization too, e.g. the Java Serializable interface.

2.1.6 Virtualization

Once the data reaches the surrogate, it is deserialized and the execution of the offloaded task is ready to start. Here comes virtualization into play. It means to create a virtualized environment -through a VM- in the surrogate to emulate the conditions of the mobile device. The VMs run as normal applications inside the surrogate's OS; this is an important feature as the different VMs corresponding to the applications of different users run separately, providing isolation and protection.

Among the offloading entities in subsection 2.1.4, the level of features does not necessarily need a virtualized environment in the surrogate side, as there might be two different versions of the same offloadable task, one for the mobile device and one for the surrogate. On the other hand, when offloading methods or images, the task in the surrogate is likely to be a copy of the task in the mobile device, and considering that the instruction set architectures (ISAs) of the mobile devices are almost always different from the ones of the surrogates, virtualization is needed. Typically the architecture of the mobile devices' processors is ARM, whereas the processors of the surrogates are x86.

There are two types of VMs:

- System VM: it emulates the ISA and functions of another machine, thus the provided ISA can be different from that of the real machine.

- Process VM: also known as application-layer VM, it supports a single process, being created when the process starts and being destroyed when the process exits. It provides a platform-independent programming environment, that usually has an associated programming language that can be compiled into object code interpretable by the VM itself. Therefore, it abstracts away the details of the underlying hardware or operating system, allowing a program to execute in the same way on any platform. Two well-known examples are the JVM (Java VM) or the CLR (Common Language Runtime, the virtual machine component of Microsoft's .NET framework).

System VMs are an acceptable approach in the context of MCO, but process VMs are of special interest as much of today's mobile devices run process VMs themselves. For example, the devices with an Android OS run the Dalvik VM, which works with Java and is very similar to the Java VM. An important difference between the Java VM and the Dalvik VM is that the first compiles the Java code into .class files written in the so called Bytecode, whereas the second produces .dex files written in an optimized object code for systems that are constrained in terms of memory and processor speed. Another example are the devices with a Windows Mobile or Windows Phone OS, most of them being able to run variations of the CLR VM.

2.1.7 Networking, mobility and fault tolerance

The process of offloading relies on wireless networks. Nonetheless, the network is not reliable by its own definition, and wireless networks are even less stable. Furthermore, the diversity of mobile networking environments and the effect of mobility increase the unreliability [Chu10]. Because of these issues, it is important to handle failures in order to provide reliable services [Clo12].

An offloading process can fail due to network congestion or failure, or server failure. If only saving energy mattered and there were flexibility to have a delayed answer, a possible option would be to wait until recovery and then offload again. On the other hand, if only improving performance were the goal, whenever an offloading process started, then it would be worth to instantaneously start

executing the task locally too, just in case a failure occurred. However, these are two extreme points of view, and while focusing on energy saving or improving performance, most MCO systems care about the overall benefits as well. Therefore, the most widely adopted strategy is to wait for a time until considering that a failure occurred and then re-execute the task in the mobile device. Recent literature investigates how should this timeout be set in order to find the optimal moment for launching local re-execution [Wan13, Mar13].

2.1.8 Infrastructures

In MCO, cloud services, PCs, specialized processors, the local environment or even other nearby mobile devices [CDA11, Par11] are candidates to be the surrogates that will execute the offloaded task. This variety arises MCO approaches that even include surrogate discovery modules, searching for the type of surrogate that fits best to execute a task with determined resource needs [Yan08].

Nevertheless, the concept of computation offloading usually involves cloud or grid infrastructures. In both cases, the infrastructure can be thought as a distributed system, sharing the resources of many computers in different locations. The machines are synchronized with workload balancers, trying to avoid saturated and under-utilized computers. The main difference is that with a grid infrastructure, the customer pays to have available a set of resources, no matter if not harnessing them. The cloud infrastructure goes one step further, providing on-demand resources as services, without the need of an advanced reservation of resources. Moreover, the cloud infrastructures offer the following desirable properties:

- Multi-tenancy: the users can share the applications in the infrastructure. Each user then runs a customized virtual application instance, but there is only a single original application. Thus, updating or maintaining an application has to be done only once, in contrast with single-tenancy architectures, where the providers need to touch multiple applications.

- Scalability: even with a large number of users, the infrastructure can allocate on-demand the resources needed for each. Balancing the workload of the cloud infrastructure has an advantage over doing it in a grid infrastructure: as the resources are not reserved, all the available resources are shared across the large pool of users. Thus, the utilization and peak-load capacity can be improved even more.

Since the amount of computation to be offloaded through MCO is not supposed to

be really large and will not always be parallelizable, the grid infrastructures are found more frequently in other works [Wol08]. On the other hand, the dynamic (on-demand) provisioning of cloud infrastructures fits with MCO like a hand in a glove.

An MCO system can adopt a public cloud infrastructure, being able to choose among many providers, such as Amazon EC2 [AWS13] or Windows Azure. Besides, there are softwares [Euc13] that help establishing an own private cloud infrastructure. However, the operational costs of a cloud infrastructure are expensive: the total cost of ownership to support increasing numbers of users can grow rapidly, the proprietary software upgrades and updates and keeping the machines on 24 hours are considerable costs. The security is also a necessary added cost; the next subsection will outline the main aspects that should be considered.

2.1.9 Security, privacy and trust

In order to keep the privacy of the users of the applications, an MCO system requires security measures both on the client side (hereafter: client side refers to the mobile device side) and on the surrogate side, as the application's data is transferred over the network from one to another.

As the network is not secure, the data must be encrypted during the communication between the mobile device and the surrogate. However, using encrypted transmissions is reflected in the performance of offloading as follows (extending *Eq. 3*):

$$\frac{I}{F \cdot M} + R + \frac{D_s}{B_u} + \frac{D_r}{B_d} + \frac{enc(D_s)+dec(D_r)}{M} < \frac{I}{M} \qquad (6)$$

where $enc(D_s)$ and $dec(D_r)$ are the amounts of computation that it takes to encrypt the data to be sent and decrypt the received data, respectively. As these jobs are done in the mobile device, they are divided by M, giving the total encryption-decryption time. In terms of energy, *Eq. 5* would also be extended accordingly, multiplying this added time by P_c. The encryption and decryption must be done efficiently, otherwise the costs would make offloading useless.

On the other side, the data will be decrypted in the surrogate. It is considered that in the case of MCO the data is needed to perform a computation with it, so decryption is needed. If the data were to be stored -MDO-, it would not necessarily have to be decrypted.

Another important security concern are the applications stored in the surrogates. The developers of the applications might not want to upload the source code of the applications to the surrogates, but rather only the compiled application ready to use. With the help of virtualization this is possible. However, either storing the source code or the applications' binaries, this data will have to be kept secure. Integrity, authentication and digital rights' management are relevant facets to consider when storing data.

2.1.10 Applicability

The last relevant criterion for the design of an MCO system is its applicability, who is going to be able to use it and with which applications.

MCO has still not evolved enough to present a totally automated system capable to improve the overall performance of the mobile devices, detecting the applications suitable for MCO, partitioning them and later offloading them. Thus, mainstream usage is not yet possible, and the target audience for the MCO systems are still the developers of the applications (many MCO systems describe themselves as middleware systems). Typically, as the offloadable entities are larger (e.g. image offloading), the system efficiency is smaller but its ease of use increases (more automation). In opposition, when they are more selective (e.g. feature offloading) the system efficiency is better but the developers' effort increases, too (less automation, especially in partitioning). However, there are offloading approaches using compact offloadable entities that propose a relatively simple framework for the developers [Kem10].

All the offloadable applications meet an unquestionable property: heaviness of computation. However, if an MCO system is not aiming for universality, it can take advantage of other traits that also fit well with the nature of MCO [CDA11]:

- Parallelizability: with a parallelizable application, the value of F in *Eqs. 2*, *3*, *5* and *6* would increase even more, since the computation could be distributed between multiple surrogates.

- Strength of expression: if the data used by the application could be translated to a more compressed format, the cost of sending data to the surrogate would reduce.

- Time flexibility: when the result of the offloaded computations is not urgent, many context-based optimizations can be done.

- State independency: allows avoiding the costly synchronization of the internal state of the application.

On the other hand, the applications with which MCO is approachable -with intensive computation- are present in many areas: multimedia processing, vision, recognition, graphics, gaming, text processing, etc. Another possibility -again, if not aiming for generality- is to focus on improving the applications of a subset of these areas.

Furthermore, the range of mobile devices suitable for MCO -with different computation capabilities- is also wide: laptops, PDAs, tablets, mobile phones, A-GPS devices, sensors, autonomous robots, etc. When designing an MCO system, it is desirable to aim for interoperability -device and location independence- and compatibility, but with this extensive offer of potential clients it is also recommendable to focus on a subset of target devices.

The last observation about the applicability of an MCO system might look redundant, but it is important to consider whether there is an interest on a system applicable in practice. For example, there are very good MCO approaches that are still only theoretical approaches because their implantation would require a network infrastructure different from the existing one, or because implementing them would require modifying a widely established mobile OS.

2.2 Related works

Although the literature about MCO is extensive and there are many papers proposing MCO systems, the MCO in practice is still in early stages and none of the proposed approaches has become of regular use.

This section presents an analysis of some of the recent MCO systems that are more referenced in the literature: CloneCloud [Chu11], Cloudlets (understood as the model in [Clo12], although it was first introduced in [Clo09] and there are other approaches based on the same concept [Gao12]), Cuckoo [Kem10], EAM (Elastic Application Model [Zha10]), MACS (Mobile Augmentation Cloud Services [Kov12]), MAUI [Cue10] and PANDA (Policy-based Model-driven Pervasive Service Creation and Adaptation [Yan08]).

Table 1 summarizes the most relevant aspects of these systems, following the

taxonomy described in section 2.1. Note that the table is divided between the aspects described in the theoretical model of the systems and the actual details of the implementation (below the row labeled publication year). As all the systems are oriented towards the model of process VMs, only the concrete VMs used in the implementation are included in the table. Also, nor serialization neither the used infrastructure are included, as all the systems require serialization and are thought for a cloud infrastructure (except for Cloudlets, as the term Cloudlet itself refers to the type of infrastructure suggested in their model). The rows labeled as client

	CloneCloud	Cloudlets	Cuckoo	EAM	MACS	MAUI	PANDA
Motivation focus	Both	Energy saving	Performance improving	Energy saving	Both	Energy saving	Performance improving
Automated partitioning	✓	✓	✗	✗	✗	✗	✓[1]
Fine-grained partitions	✓	✗	✓	✓	✓	✓	✓[1]
Dynamic decisions	✓	✓	✗[2]	✗	✓	✓	✓
Offloadable entities	Images	Images	Features	Features	Methods	Methods	Methods
Fault tolerance	✗	✓	✓	✓	✓	✓	✗
Security measures	✓	✓	✗	✗	✗	✗	✗
Publication year	2011	2012	2010	2010	2012	2010	2008
Client Platform	Android[3]	Android	Android	Any	Android	Windows Mobile	Windows Mobile
Surrogate's VM	Android x86	KVM	JVM	-	JVM	CLR	JVM
Ease of use	+++++	+++	++	++	++	+++	++
Available	✗	✗	✗	✗	✗	✗	✓

1 It identifies the original Java classes of the application with more offloading potential. Although this partitioning procedure is done automatically, the application must be carefully designed beforehand to take advantage of PANDA.
2 It is able to execute the application locally if there is no network connection, but otherwise it decides to always offload, regardless of the dynamic conditions.
3 With a modified Dalvik VM.

Table 1: Analysis of recent MCO systems.

platform, ease of use (for the developers) and availability gather facets of the applicability of the systems. Observe also that it is always desirable to save both energy and time, but the row labeled motivation focus refers to which of them the system emphasizes more.

In general, it is reprehensible that the majority of the systems (except Cloudlets) assume the network to be stable, and the local re-execution strategies that they adopt in case of failure are very simple. Similarly, security is not the object of attention in these approaches. The systems reviewed take advantage of the facilities that the process VMs provide, focusing on mobile OS like Android (with the Dalvik VM) or Windows Mobile (with CLR) that already include their own VM. However, it would be possible to implement virtualization at the level of a system VM, offering MCO to mobile operating systems like iOS. It is worth to mention that in general, as a system becomes more automated -reducing the burden on the applications' developers that are going to use them- it becomes more sophisticated, too, like CloneCloud. The efficiency of a system is also reduced as the level of automation grows.

Interestingly, the automation of CloneCloud is possible due to an architectural design that differs from the others: the mobile device has a copy of its contents (the clone) in the cloud. This way, the heavy computational tasks that can be offloaded can include operations such as file searches, virus scans, image searches by content (CBIR), etc. In practice, this approach needs using MDO continuously to have the clone always synchronized with the original, and even if done in background, this consumes energy. On the other hand, MAUI focuses on saving energy, and their decision-making system (called MAUI Solver) is found in the surrogate instead of the mobile device. When a potentially offloadable task is to be executed, it is invoked asynchronously from the mobile device. Avoiding the waste of energy that would produce calculating the best decision is a good idea; however, waiting for it to be calculated in the surrogate can be a waste of time.

In contrast with CloneCloud, systems like Cuckoo, EAM, MACS, MAUI or PANDA involve more adaptation by the developers of applications. Their approaches differ a lot though. Cuckoo, EAM and MACS require the developer to specifically design the application following certain patterns. Cuckoo proposes a framework where the potentially offloadable tasks must be implementations of a Java interface defined through the Android Interface Definition Language (AIDL), allowing for two different versions of the same task, one to be executed locally and the other remotely (MACS also allows two different versions). Similarly, with EAM the potentially offloadable tasks must be encapsulated in independently

runnable parts of software called *weblets*. On the other hand, MAUI and PANDA bet for more simplicity of use. Both MAUI and PANDA work with the original application, the first working with methods and the second with classes. MAUI decides dynamically whether to offload a method or not (actually, among a subset of methods marked as remoteable by the developer) and PANDA takes the decisions at class level. However, both systems will not be useful if the original design of the application does not place the heavy computation parts separately. The systems working with features or methods are usually stateless, every computation request to the surrogate is independent from previous or later requests.

Some results presented by these systems are a computation speed-up by a factor of 60 and reduction of battery consumption by a factor of 40 with an object recognition application thanks to Cuckoo, energy savings of 27% for a video game and 47% for a chess game thanks to MAUI or factors up to 20 both in energy saving and performance improving by CloneCloud. The authors of MAUI also report that they doubled the frame refresh rate of the video game with their system; however, this is only possible if the application has flexibility, meaning that it does not need to have the result of the offloaded computations instantaneously (the application can keep the execution and the result of the offloaded computations will be asynchronously processed, when it arrives).

Apart from the analyzed systems, there are a number of other systems related with partitioning, migration, and with MCO in general: AIDE [XGu04], AlfredO [Giu09], Amoeba, [Mul90], ASIMS [CAi11], Chroma [Bal03] (based on Spectra[Fli02]), Dessy [Lag11], Hydra [Wei08], MCM [Flo13], Odyssey [Nob97], OLIE [XGu03], Potrium [You01], Scavenger [Kri10], Slingshot [YSu05], Sprite [Dou91], Wishbone [New09] among many others.

3 The proposed system

This chapter presents the main work of this thesis: a proposal of an MCO system. First, the theoretical approach of the system will be presented. AESTET is a subpart of the MCO system, but has its own section in 3.2. The next section reveals the most remarkable implementation details of the system. Section 3.4 presents many results as well as an evaluation of the system, discussing its limitations. Finally, the last section points out some directions for future research.

3.1 Theoretical approach

To describe the chosen design of the system, the next sections will go through the main aspects reviewed in the taxonomy presented in section 2.1. However, the motivation (2.1.1) and the applicability (2.1.10) considerations of the system will be included in the general description (3.1.1), since these aspects are important to understand the decisions taken. The partitioning (2.1.2), infrastructure (2.1.8) and security measures (2.1.9) will be included in the second subsection (3.1.2).

3.1.1 General description

The general behavior of the system will be outlined in this section. First, it is worth to clarify that the system is designed (in collaboration with [Mar13]) according to the questions that this thesis wants to answer. Thus, some of the aspects of an MCO system reviewed in 2.1 will not be considered or simplified here, as they are not necessary for the scope of this approach. Given that the reviewed MCO systems are either not available or their objectives differ from the ones in this thesis, the system presented in this thesis is not based on any previous approach. This is another reason of the limited scope of the system, as due to time constraints, this thesis does not allow the unfolding of a new and complete system. This is acceptable, since another aim of this thesis is experimenting and providing a base system able to be extended in the future.

The system focuses on improving the performance of the mobile devices' applications, although aiming to save as much energy as possible at the same time. This choice will be explained in detail in 3.1.3. The system is mainly thought for mobile phones, more concretely, for smartphones. Furthermore, the concept of the system will be similar to the majority of the MCO systems, it is designed to act as a middleware between the applications' developers and the functionalities of MCO.

It has been explained that MCO appears mainly in areas like multimedia processing, vision, recognition, graphics, gaming, or text processing, where heavy computation tasks are performed. It can be observed that the heavy computation tasks of these applications have significantly variable execution times depending on its input (how big is the image where faces must be detected, how long is the audio file where the speech must be recognized, which is the difficulty level of the artificial intelligence, etc.). Furthermore, these tasks often behave similarly to deterministic algorithms. A deterministic algorithm is an algorithm that, given a particular input, will produce always the same output passing through the same sequence of states (on a given computer, taking approximately the same amount of execution time). In this thesis, it is assumed that the computationally intensive tasks suitable for MCO will not necessarily produce the same output given the same input, but it will be assumed -and this is the main basis of the system presented here- that they will take the same execution time given the same input. Thus, it is possible to predict the execution time of these tasks given a particular input.

In section 3.1.3, it will be seen that the main feature of the system is its ability to take dynamic offloading decisions. The system will consider that it is worth to offload a task when its estimated execution time on the mobile device is greater than the sum of its estimated execution time on the surrogate plus the predicted costs of the data transfers. These estimations will be possible assuming the behavior described above. AESTET will take care of that job. Although this system is intended to be an automated system, it will be seen that it expects the developer of the application to provide a particular function on development time. This consideration is important now because it influences the design of the system.

The offloading decisions of the system will be taken at runtime, just before the execution of a potentially offloadable task. This ability is distinctive because it permits the system to handle tasks dealing with real-time data (e.g. a face recognition application that uses as input a newly captured picture), in contrast with other MCO systems that assume the data already offloaded in the cloud. Besides, the type of tasks suitable for this system must be a module, independent from the rest of the application, as the system will not keep a state between the mobile device and the surrogate. The rest of properties explained in 2.1.10 (parallelizability, strength of expression, time flexibility) are not considered.

The system offers interoperability through an abstraction layer from the network. It is dynamically taken into account the type of connection used by the mobile

device, e.g. 3G, Wi-Fi, LTE, etc. predicting more delay in the corresponding cases.

3.1.2 Architecture

The system will not be able to partition an application automatically. First, because AESTET requires the developer to provide a particular function for each potentially offloadable task on development time -as mentioned above-. Second, because it is out of the scope of the thesis. Instead, it is expected that the developer of the application does the partitioning beforehand. The system is oriented towards applications written in the Java programming language, and the way to partition the application will be packing the computationally intensive parts inside a JAR (Java ARchive) file with a few peculiarities (further details can be found in [Mar13]). Thus, the offloadable entities of the system are considered features.

The potentially offloadable tasks will have to be uploaded to the surrogate in advance, too. The system provides a web interface tool that facilitates this task [Mar13]. The developer also has the option to implement two different versions of the task, one for the mobile device and one for the surrogate. This tool allows to upload a copy of the original task[5] of the mobile device as well as an alternative implementation. Having the tasks already uploaded to the surrogates when the applications start to execute avoids having to upload them during runtime, which is a very costly approach. On the other hand, the procedure requires more burden on the developer.

The system would ideally be used with a cloud infrastructure, but for scope limitations the surrogate will be considered a single server. Note that with a single server, it makes sense that the developer uploads in advance the potentially offloadable tasks, as it would not be a good approach to do it automatically from the clients. If this were the case, all the users using the same application would check its existence in the surrogate, and if the application were not there, it would be automatically uploaded. The problem of this approach is that only the first user doing this check would actually upload the task, as the rest would already find it. In contrast, this approach would be more acceptable in the context of a cloud infrastructure, where there would be multiple servers and doing this checks from the client side would not be useless. Regarding this possibility, it could be argued that without the need of the developer uploading the tasks to the surrogate beforehand, the system could gain even more automation -eliminating all the interaction with the developer- with auto-partitioning. However, automatic

5 Sometimes an exact copy will not be possible, as explained in 3.1.4.

partitioning is not possible because of the features of AESTET.

The system is basically divided into two parts. The client side and the surrogate side. On the client side, the system is just a small software library that can be added to the applications. After adapting the application appropriately to make use of the MCO functionalities offered by the system, it will automatically improve the application's performance deciding dynamically the best option – execute a task locally or offload it. It will take care of the communication with the server, sending the request and handling the response.

On the surrogate, the system is built on top of a web server. This choice is initially taken as a simplification decision. It provides many commodities, the virtualization -as will be seen in 3.1.4-, communication -listening and handling the requests- and security are much simpler taking advantage of it. However, the facilities of object serialization and transmission that provide technologies such as Java RMI or CORBA are not exploited. Thus, the client and the surrogate will communicate

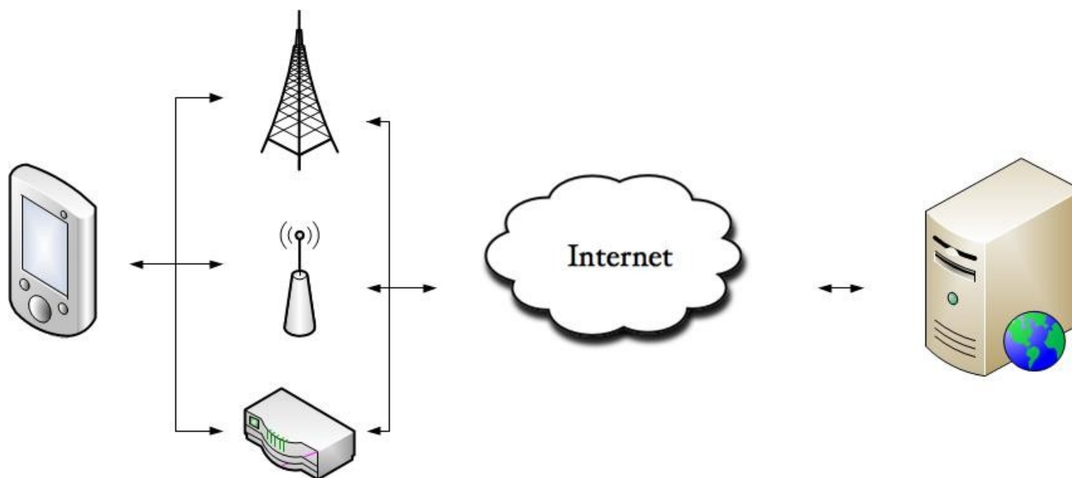

Figure 2: Architecture of the proposed MCO system. The mobile phone can access the surrogate through cellular towers (for cellular networks like 3G or LTE), through WAPs (Wireless Access Points, for Wi-Fi networks) or by other means.

through the HTTP protocol. More concretely, the HTTPS protocol is used to provide a minimum grade of security, encrypting the exchanged data through SSL. The resources in the surrogate are not protected against untrusted users with an user identification system. However, the execution of a task can only be queried using the identifying name of the task, which cannot be known by untrusted users unless reading the contents of a request (which cannot be done thanks to the

encryption). The small overhead produced by the encryption will be considered part of the data transmission costs in this thesis. Figure 2 illustrates the architecture of the system.

3.1.3 Decision-making

The system was originally designed to take into account both the energy saving and time saving criteria. The idea was to put together the *Eqs. 1* and *4* (seen in 2.1.3) into one single criterion. Basically, the two criteria indicate almost always the same decision. For the few cases where they differ, there were two possible approaches. First, give more weight to one criterion or the other depending on the remaining battery of the mobile phone, as with low battery it might be preferable to save energy and with more battery it might be preferable to give more emphasis to performance. However, this is not objective for all the cases. A user might prefer to improve performance even if the battery level is very low, as he is home and can load the phone whenever needed. If another user is traveling, she might want to save battery even when the mobile device is fully charged, as it could be difficult to find a place to reload it. Considering this, the second approach was to take into account the user preferences to decide which criterion should be given more weight.

As seen in 2.1.3.1, to evaluate the energy saving criterion, the values of P_i, P_c, and P_t must be obtained from the phone. However, it was pointed out that the mobile phones of nowadays cannot provide this information, and the idea was discarded.

Thus, the dynamic decision algorithm of this system is based on the time saving criterion (*Eqs. 1*, *2* and *3*). Nevertheless, this thesis also wants to prove that by improving performance energy is saved. A trivial case that shows that, is when the data to be offloaded is almost negligible. Then, referencing again the *Eqs. 1* and *4* (seen in 2.1.3), it would be considered that $D=0$. The resulting inequalities would be as follows:

$$\frac{I}{S} < \frac{I}{M} \tag{7}$$

$$P_i \cdot \frac{I}{S} < P_c \cdot \frac{I}{M} \tag{8}$$

It is easy to see that when the first is true, the second is also true, as $P_i < P_c$. However, these equations only prove the case when D is negligible (in addition, here the RTT is not considered). Hence, in the section of evaluation other cases will be studied practically: although the sensors of the mobile devices are not

precise enough for fine power consumption measures, there are some external devices [PMo13] and software applications [Yoo12, PTu13] able to measure the power accurately.

The decision-making procedure of the system is done at runtime, just before a potentially offloadable task is to be executed. The execution of this decision-making procedure is done in the mobile device, in contrast with other systems [Cue10] that do it in the surrogate. The advantage of doing it outside, is that the

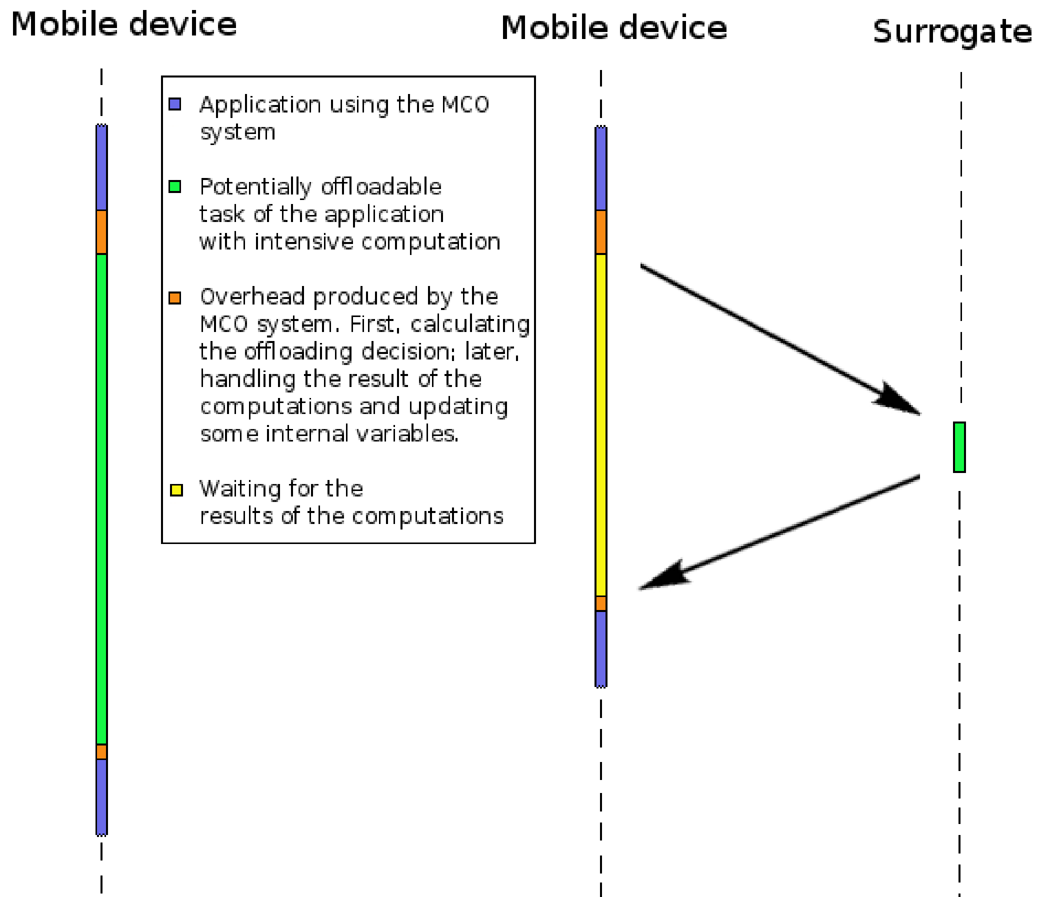

Figure 3: Two executions of an application with a potentially offloadable task that uses the MCO system presented in this thesis. On the left, the task is executed locally; on the right, it is executed remotely.

overhead (both in terms of time and energy) of the calculations is much lesser, as the surrogate has a computing capacity far superior than the mobile device has. The disadvantage is that the RTT needed to transfer and retrieve data from the

surrogate might take longer than the calculations themselves. If the focus is saving energy, computing the decision-making procedure outside the mobile device might be feasible, but when aiming at performance improving, the delay of the RTT might be too much.

Thus, it will be very important to design an efficient decision-making system, otherwise the approach of calculating the decision in the surrogate would have been better. Furthermore, too much overhead would not compensate the benefits of the system. Figure 3 shows two cases of the execution of an application using the system.

In line with what has been said above, the dynamic offloading decisions of the system are taken following *Eq. 3* of 2.1.3. However, the system is not designed to make predictions over the size of the responses (D_r). Instead, it will be assumed that the returned data is always a small set of data, thus negligible. Considering this, the inequality in *Eq. 3* can be rewritten as follows, in order to represent the dynamic offloading decision criterion of this system:

$$\frac{I}{F \cdot M} + R + \frac{D_s}{B_u} < \frac{I}{M} \tag{9}$$

3.1.3.1 Parameters

This subsection describes how the system gets the necessary parameters to make an offloading decision, i.e. how the system evaluates the inequality shown in *Eq. 9.* Observe that the problem of evaluating *Eq. 9* is equivalent to forecasting the following:

- Data transferring time: it is the sum of the time R -which mainly represents the RTT- plus the division D_s / B_u. Many papers combine these two parameters into a single one. However, the focus of this thesis is to be able to take the correct offloading decision for heavy computation tasks with significantly variable execution times depending on the properties of their inputs. If the size of the inputs were always relatively big, a very precise prediction of these times would not be necessary. However, as the system will deal with inputs of different sizes, it might encounter cases with small inputs, thus needing more accuracy of prediction. Other works suggest the importance of separating the RTT and the bandwidth, too [Cue10].

- Estimation of the task execution time on the mobile device: corresponding to the right side of *Eq. 9*: I/M.

- Estimation of the task execution time on the surrogate: corresponding to the addend in Eq.9.: $I/(F \cdot M)$.

The randomness of the network state makes the forecasting of the network parameters especially complicated. Bayesian probabilistic approaches are suggested [Wol08], but this produces too much overhead for the purposes of this thesis and a rough estimation method will be used instead.

The estimation of the task execution time must be predicted both in the mobile device and in the surrogate. Two methods are proposed:

- Direct cost function: initially, the system expected the developer of the application to provide a cost function of his potentially offloadable tasks. This means, a function capable to estimate the amount of computation I of a task, given an input.

- AESTET: This system will be explained in detail in section 3.2.

In both cases though, the computation speed relation F between the mobile device and the server is needed. In the first case, the cost function provided the value of I in the equation, therefore F was still needed. The need of F in the second case will be explained later in 3.2.

Having directly a cost function is of course more efficient than AESTET, but this option was too unusable (many developers would not be able to find such a function), and AESTET was designed.

Considering that the value of the computation speed of the surrogate S is a known value, and that $M = S/F$, calculating F is enough to know M. Then, the parameters that the system needs to calculate for the evaluation of the decision criterion are: F, I, R, B_u. and D_s. The system will make the following categorization for these parameters:

- Persistent parameters: F.
- Parameters depending on the execution environment: R and B_u.
- Parameters depending on the input of the task: I and D_s

The persistent parameters only need to be calculated once, as they will stay the same always (the mobile device is always F times slower than the server). F is calculated in the first execution of an application using the system, and then is stored persistently. It can be retrieved then in the future, avoiding the need of recalculating it.

The parameters depending on the execution environment are actually the network dependent parameters. Different strategies were tried in order to calculate these parameters. A complex probabilistic prediction system of the state of the network would produce too much overhead, so the approach was simplified. The parameters could be readjusted continuously during the execution of the application measuring background queries to the server, but this would affect the performance and would consume energy. Considering the case of cellular networks like 3G, the randomness of these parameters is high, so it would be still not reliable to recalculate them continuously. When using a Wi-Fi network, while staying under the coverage zone, the network state is quite stable, so again, there would be no point in recalculating the parameters continuously. Hence, it was decided to calculate these parameters only once, at the beginning of the execution of the application.

The parameters depending on the input of the task can only be evaluated immediately before the execution of the task.

The system is designed to calculate all the parameters from practical observations. The following list reveals the details for each one:

- F: a simple algorithm (which mainly does some empty loops) with a known execution time cost on the surrogate is executed on the mobile device during the first execution of the application, and its execution time is measured. Comparing its execution time in the mobile device and the one in the surrogate, F is obtained. It is very important to note, that the moment when this algorithm is executed is very relevant. The OS of a mobile device might assign more computation resources to a process at different moments of its execution. This is studied in [Mar13], and summarizing, the best approach is to calculate F just when the potentially offloadable task were to be executed. Besides, observe that the iterating test algorithm proposed to obtain the speed relation might be able to give a relatively accurate value to F. However, another important note is that the behavior of the mobile device might be different for different tasks. This means that comparing the execution times in the mobile device and in the surrogate of another unknown computationally intensive task, the value of

F could vary slightly. As the accuracy is crucial for the purposes of this thesis, the solution offered by this system is to maintain different computation speed relations (a set of F's) for each potentially offloadable task of an application. These computation speed relations are also stored persistently. Furthermore, as the application will keep being executed by the user, each time that a potentially offloadable task is executed, the value of the computation speed relation for that task will be updated (the updated value will be an average of the past values and the newly obtained one).

- R: at the beginning of the execution of the application, the surrogate is queried a few times (currently 10) and the RTTs are measured and averaged. If the connection is lost or changes to another network (e.g. from 3G to Wi-Fi), R is going to be recalculated.

- B_u: inspired by [Cue10], at the beginning of the execution of the application a small file is sent to the surrogate to get the data transferring speed. If the connection is lost or changes to another network (e.g. from 3G to Wi-Fi), B_u is going to be recalculated.

- D_s: this variable can be immediately evaluated. Once the input of the task is known, its size can be obtained directly. The system is working at the level of features, but the way to handle the tasks is actually through method calls. Thus, just before the execution of the task, the system has all its its input parameters and can easily calculate the sum of their sizes.

- I: as mentioned above, a first approach was to calculate this variable using a cost function provided by the developer of the application. When using AESTET, I is not needed.

The practical way in that the system calculates the different variables is vulnerable to some unfortunate circumstances. For example, if many applications are being executed in parallel while F is being calculated, the obtained value might be not accurate. However, as F will keep being updated, this is not a big problem. Another example would be that other applications could be using the network while R or B_u are being calculated. This could affect the behavior of the system during one single execution, but as R and B_u are going to be recalculated for each execution, this is also not unacceptable.

3.1.4 Serialization, virtualization and fault tolerance

The serialization step is not provided by the system, due to scope limitations. The system transfers already serialized parameters to the surrogate, and the developer is expected to implement this serialization. The virtualization of the system is implicit thanks to the JVM. Both the client and the surrogate use a JVM. The system handles fault tolerance in the same simple way that other systems do: if the offloaded execution of a task fails, the system re-executes the task locally [Mar13].

3.2 Automated estimation system of task execution times

This section presents AESTET, the statistical estimation system used to automatically forecast the execution times of the potentially offloadable tasks of an application.

3.2.1 Overview

The statistical scheme of AESTET (inspired by [Ive96]) is based upon a nonparametric regression technique. With it, the execution times of the tasks are forecast from past observations. This follows the philosophy of the MCO system in 3.1: to obtain the parameters of the system from practical samples. Furthermore, this approach avoids having to understand the computational complexity of the tasks, in contrast with other techniques such as static code analysis [Rei94].

The technique used by AESTET is able to compensate for different parameters upon which the execution time depends, which is important for a dynamic decision MCO system. Moreover, it does not require any knowledge of the architecture of the target machine, making it suitable for any platform. Another important feature is that AESTET is almost not affected by the presence of outliers -erroneous data- in the set of observations.

However, this type of system fits better for the case of distributed computing. In this case, huge tasks must be executed and a scheduler must estimate their execution times in order to decide where to execute them. As the tasks are large, the overhead produced by computing the estimations is not relevant. Furthermore, in many cases it is not urgent to produce the estimations, but when using this system for MCO the results must be immediate. Because of this, when using this estimation system in contexts other than MCO, it is not an obstacle to deal with tasks with multiple input parameters. The nonparametric regression technique described in 3.2.3 requires finding “similar cases” among the previous stored

observations, and the cost of this operation increments dramatically as the input of a task is more complex (e.g. more parameters). As said, this is acceptable in the context of distributed or grid computing, because the overhead does not matter too much. But in the case of MCO, this is not acceptable. The estimations must be done efficiently, with a low overhead. Thus, AESTET requires the developer of the application to provide translation functions that summarize the input of his potentially offloadable tasks into a single numeric value. Hereafter this will be referred as input representation. This input representation mainly needs to satisfy one condition, inputs that lead to similar execution times should have similar representative numeric values. The nonparametric technique can still produce acceptable results if the input representation is good, for example, the authors of [Ive96] use the size of the input as the input representation, and affirm an error rate smaller than 20% in the estimations produced.

As AESTET needs past observations to produce estimations, it needs some samples before starting to work. In the tests shown in [Ive96], it is said that around 10 initial samples are enough. However, this is only enough if the tasks have an execution time strongly correlated with the input representation. As the system cannot totally rely on the accuracy of the translation function provided by the developer, a bigger initial sample set is needed.

Two options were considered regarding the creation of this initial database. First, the application could go through a training phase, where AESTET would be only gathering samples. Once the database would reach an enough big size, AESTET would be able to start producing estimations. The second option is that the developer generates this database beforehand (in development time) and includes it within the application. The first option would make the things easier for the developer, and AESTET would gain usability, but all the users using an application with that system would have to go through this training phase. In contrast, with the second approach the generation of the initial database has to be done only once by the developer. Thus, the second approach is chosen, and an online database generation tool [FUB13] is provided to give facilities to the developers.

In order to generate the initial database through the web tool, the developer must provide a set of sample inputs of his potentially offloadable tasks. Although this is again more burden on the developer, it is much more precise than using random input generators [Chu11].

3.2.2 Database design

Once an execution is done, an estimation is computed from the previous values stored in the database that was added to the application. After the execution, the real execution time is saved to the database, either if the task has been executed locally or in the surrogate. Thus, in the database will coexist execution times produced in the mobile device and execution times produced in the surrogate. Using the parameter F described in 3.1.3.1 it will be possible to convert from execution times produced in the surrogate to local execution times, and vice versa. Because of this, the database needs an extra attribute for each entry indicating whether it was produced in the mobile device or in the server. Then, the entries of the database will have the following attributes: <*inputRepresentation*, *executionTime*, *isLocal*>, being *inputRepresentation* an integer number, *executionTime* a real number and *isLocal* a boolean attribute.

Each different task will have its own table in the database. An index will be created for each table, to keep them sorted by *inputRepresentation*. This way, querying the database will be faster thanks to the index and the system will be able to produce the estimations more efficiently. However, once the execution of a potentially offloadable task is done and the newly obtained real execution time is to be added to the database, the insert statement will take longer because the index will have to be updated at the same time. As this can be done in background after the execution of the intensive task, it will be not considered a problem.

3.2.3 The nonparametric regression technique

The regression technique used for the execution time estimation problem in this thesis is based upon a technique known as k-Nearest Neighbor (k-NN) smoothing. AESTET adapts this technique and follows the steps of this algorithm:

1. Once a potentially offloadable task T is to be executed, its input is known. The system obtains the input representation through the translation function provided by the programmer.

2. The system searches in the table of the database corresponding to the k nearest entries of T by *inputRepresentation*.

3. The k elements found are retrieved. For each of them, the attributes *executionTime* and *isLocal* are obtained. If *isLocal* is true for an entry, the *executionTime* is divided by the current F (the computation speed relation

between the mobile device and the surrogate, explained in 3.1.3.1) of the task. This way, the set of k elements is normalized to exclusively surrogate's execution times.

4. This set is smoothed: the execution times are averaged, and the execution times too distant to the average (outliers) are eliminated from the set. If there are one or more elements with an *inputRepresentation* equal to the input representation of T, a new average execution time *AvgSurrogate* is calculated among these and the algorithm jumps to step 6.

5. Among the resulting set, a new average execution time *AvgSurrogate* is calculated, giving more weight to the execution time of the elements with an *inputRepresentation* closer to the input representation of T. The weights are given through a weighting function (also called a kernel function) known as Epanechnikov Kernel [Ive96], which has certain optimality properties. This technique also permits to calculate the execution time of cases such that the input representation of T is out of the boundaries of the existing input representations of the database -i.e. it is not between any two other existing input representations-.

6. The average execution time *AvgSurrogate* will be the execution time estimation for the surrogate predicted by AESTET. Multiplying *AvgSurrogate* by F the value of *AvgLocal* is obtained, which will be the predicted execution time estimation for the mobile device.

7. Once the execution of T is completed, the real execution time (either in the surrogate or in the mobile device) is added to the database. If there were already 20 entries with the same *inputRepresentation* that T has, one of them would deleted randomly before adding the new execution time. This was decided to keep the size of the database not too large.

The secret of this technique resides in deciding which is the appropriate number of nearest neighbors -k- to initially search for in the database. If k were too big, the average would include too much values and would not be precise. On the other hand, if k were too small, only a few elements would be considered to calculate the average, which would increase its randomness, considering that there might be outliers in the database.

Given a total number of entries n in the database, studies have shown that k should increase in proportion to $n^{4/5}$. In the case of this thesis, testing has shown

that an appropriate assignation is $k = n^{4/5} / 5$. Thus, the computational complexity of the algorithm is $O(n^{4/5})$. The complexity helps getting an idea of the overhead that the estimations of AESTET will produce, but testing must be done to check it precisely (evaluation of the overhead is done in section 3.4).

The effect of this technique when calculating weighted averages is similar to doing linear interpolation. This means, if a task has an execution time cost that grows approximately linearly as its input representations grow, the estimations produced will be good. With higher polynomial growths (quadratically, cubicly, etc.) or exponential growths, the estimations will be worse but still acceptable for the purposes of this thesis.

3.3 Implementation

The MCO system proposed in this thesis is built extending a basic offloading engine [GM13]. The implementation of the engine in the client side is carried out for the Android platform, which fits very well with the approach of this thesis, as it works with Java. The same applies for the surrogate side, which will be running a Tomcat Server, that also works with Java. At the moment, there is only one server [FUB13] working as the surrogate.

Although Android uses Java, it is not running exactly a JVM. Instead, Android has its own Dalvik VM. It shares a lot of core functionalities and libraries with the JVM, but it is designed to cope efficiently with constrained resources and has its particularities. This will carry some compatibility problems, as will be seen in the next section.

As explained when describing the proposed system, the communication between the client and the surrogate will be done through HTTPS queries. More concretely, the query strings will follow this format:

https://www.mi.fu-berlin.de/offload/run?algName=nameOfTheTask¶m1=valueOfParam1¶m2=valueOfParam2&...

where *nameOfTheTask* is the identifying name of the computationally intensive task to be executed. The tasks are launched through a single method call that needs all the input parameters; the rest of the query string are these parameters. The server can be queried either via GET requests, as shown, or with the equivalent POST requests. The client uses POST requests to query the server

because the input parameters can be of any size, and a GET request would be restrictive. The server will answer with XML formatted data that the client will be able to parse and interpret.

In the case of the client, the software is a small set of Java classes that a developer can add to his application in order to enjoy the MCO functionalities proposed in this thesis. The functionalities of the classes are the following:

- Engine: the core class of the system. The applications only interact with this class, requesting the execution of their potentially offloadable parts whenever needed. This class will take the offloading decision, and will be the responsible to proceed with a local or remote execution. In case of a remote execution, it will handle as well the communication with the surrogate.

- Algorithms: called from Engine, it is a wrapper to the potentially offloadable parts of the application. The developer needs to adapt it.

- DataBaseHelper: in case the developer does not provide a cost function of the execution time for the potentially offloadable parts of the application and therefore AESTET is being used, this class manages the database from which AESTET takes the past observations in order to make the forecasts.

In the case of the surrogate, the software is designed to run in the Tomcat Server as a web application in the WAR format (Web application Archive). Mainly, the software serves the queries of the clients, executing the requested computationally intense task in each case and answering the client with the results of the computations. Moreover, the web application offers a WUI [FUB13] (Web-based User Interface) with two tools for the developers of the applications:

- JAR uploading: as described in 3.1.2, the developer must partition the application manually, packaging the computationally intensive parts of the application into JAR files. This tasks must be placed in the surrogate side before the distribution of the application is started. This tool provides a simple way to do so, and updates the system in the surrogate side to be aware of the newly uploaded task.

- AESTET database generation: as described in 3.2.1, AESET requires the developer to generate an initial sample database beforehand. This can be done through this tool. After the generation process, the database can be

downloaded and must be added to the application's resources. The system generates a SQLite database, since this is the format that the Android systems can handle best.

The WUI provides the necessary instructions for the use of these tools. There is a user authentication system to access the management area where the tools are located. However, right now it is limited to one user, as the software does not implement the necessary isolation and different users could modify the computationally intensive tasks uploaded by others.

There can be found also in the WUI the source code of the client and the surrogate sides, and the source code of two example applications (used next in the section of evaluation) that show the capabilities of the system.

3.4 Evaluation

In order to evaluate the MCO system, the described implementation has been used for the experiments. This section shows the results of these experiments and their interpretation.

3.4.1 Experiments' setup

Many real applications have been considered to test the MCO system. However, the implementation of the system only allows for computationally intensive applications written in pure Java. Many of the applications that were tried, had either native code calls (architecture dependent) through the Java Native Interface (JNI) -e.g. [Sph13]-, or code using some Dalvik libraries that are not present in the JVM, -e.g. [Jav13a]-.

Thus, an example application called EngineTesting was prepared. The application includes many computationally intensive algorithms, and uses the MCO system to decide whether to execute them locally or remotely. A chess game [PCA13] is adapted to the system as well in the work of [Mar13]. Here, the most relevant results are analyzed.

Although the system presented in this thesis allows for different versions of a task in the mobile device and the surrogate, all the tests have been done using an exact copy, as for throughput comparison this was the best option.

All the tests have been done using a Wi-Fi network, as none of the devices used for the tests had a cellular network (like 3G) connection available. In order to test the forecasting capacity of the network parameters, the high instability of 3G would have been interesting though (studied in [Mar13]). Fortunately, for the testing of the prediction capacity of task execution times, the type of network used is not relevant.

At the surrogate side, a server of the Freie Universität Berlin is used. The server processes with 4 cores of the type Intel Xeon CPU E5649 2.53 GHz, with a main memory of 7786 MB. The server runs Apache Tomcat 6 and uses Java 1.6. At the moment of the tests, the distance to the server was always the same (medium distance, about 100 milliseconds of RTT).

At the client side, the mobile device was a Samsung Galaxy Nexus, with a processor of 1.2 GHz, Dual Core. The server is about 30 times faster than this device ($F = 30$, according to the notation of 3.1.3.1).

3.4.2 Results

This subsection presents the results of the tests, which will be discussed in 3.4.3. The figures from 4 to 7 show the forecasting capabilities of the network parameters (RTT and bandwidth) of the system. In these tests, the client executes a task with inputs of different sizes: 100B in Figure 4, 10KB in Figure 5, 250KB in Figure 6 and 1MB in Figure 7. The task to be executed is a simple iteration algorithm with empty loops, that can be adjusted to do as many iterations as desired. This way, the task is executed many times, each producing different amounts of computation. The X axis of the charts corresponds to this amount of computation (in millions of iterations), and the Y axis to the time that it took the execution (in milliseconds). These figures show the relation between the data to be sent D_s and the amount of computation to be done I, and how the system can predict the most beneficial decision for different cases. In these tests, the amount of computation I is not predicted using the AESTET system, as the purpose of these figures is to show the network forecasting capabilities of the system rather than the execution time prediction accuracy.

On the other hand, figures from 8 to 10 show the execution time prediction capabilities of AESTET and the overhead that calculating the predictions produces. In Figure 8, the meaning of the axes is the same as the figures 4 to 7. The X axis of the figures 9 and 10 also corresponds to different inputs, like the previous figures. However, they are not ordered by computation amount like in the

previous figures. Instead, as the computationally intensive task represented in these figures is the artificial intelligence of a chess game, the inputs (the situation of the board) are ordered temporarily in the same way as they were produced while playing the chess game (the first 20 moves are shown).

No results of energy consumption are presented, as external power measurement devices like Power Monitor [PMo13] were not available during the testing. Instead, energy consumption software solutions were tried: App Scope [Yoo12] and Power Tutor [PTu13]. These applications use previously created models to predict the energy consumption and are still in alpha stages. App Scope only supports the Galaxy One mobile device and Power Tutor cannot estimate the energy consumption of the Wi-Fi adapter of the device used for the tests (Galaxy Nexus). In order to study the energy consumption, [Mar13] presents an analysis based on common values of P_i, P_c, P_t.

All the figures are produced averaging 5 execution times for each value. The execution times (its improvement) are used as the metric for evaluation of the proposed MCO system.

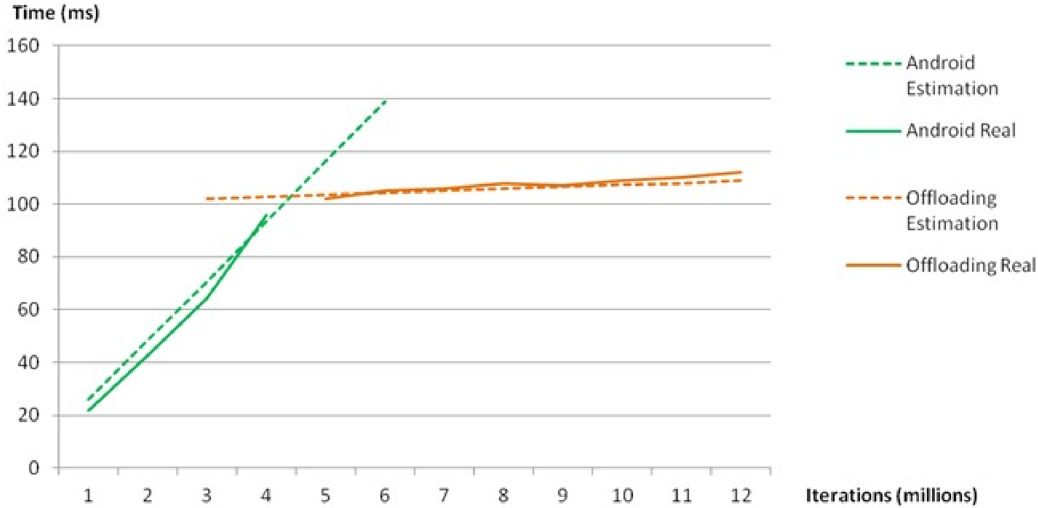

Figure 4: Behavior of the system with different amounts of computation, with 100B of input data of the task.

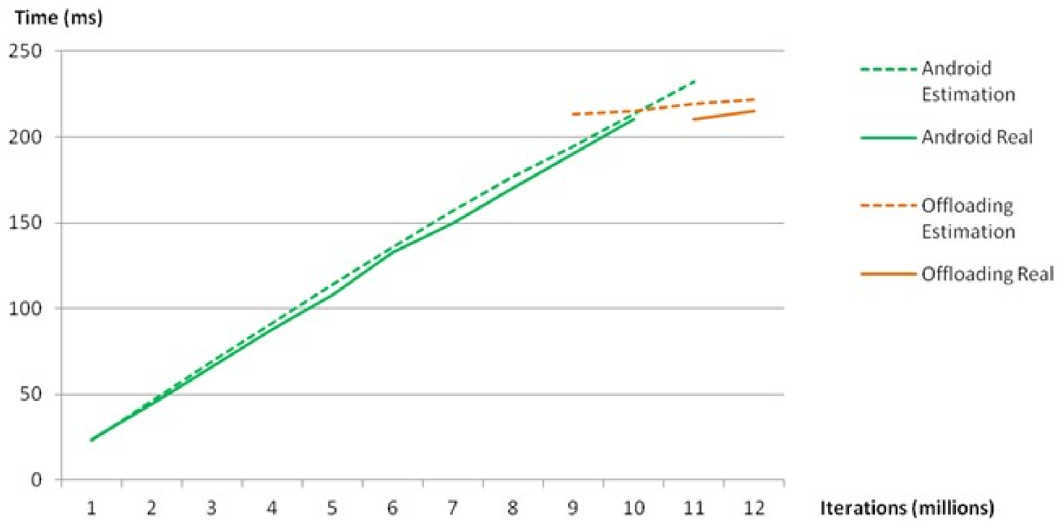

Figure 5: Behavior of the system with different amounts of computation, with 10KB of input data of the task.

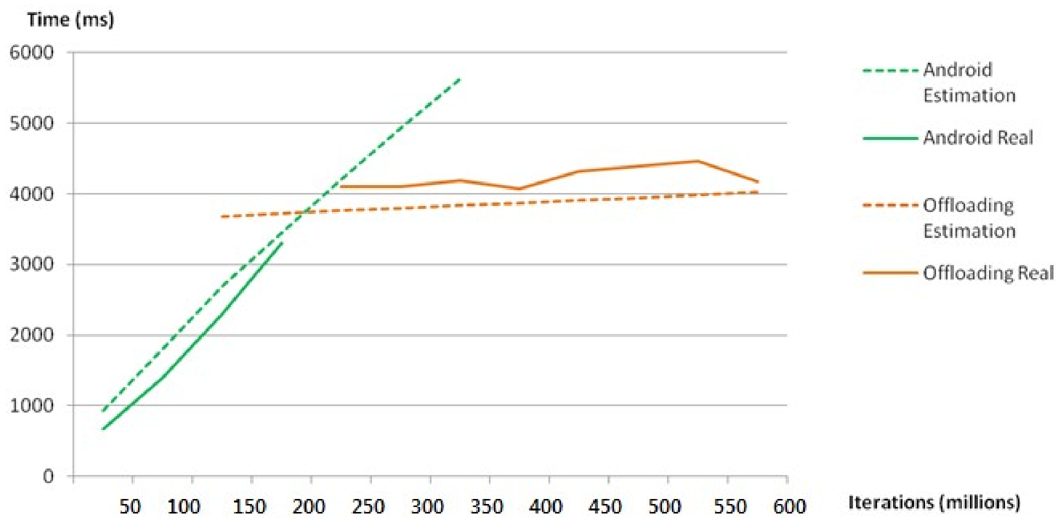

Figure 6: Behavior of the system with different amounts of computation, with 250KB of input data of the task.

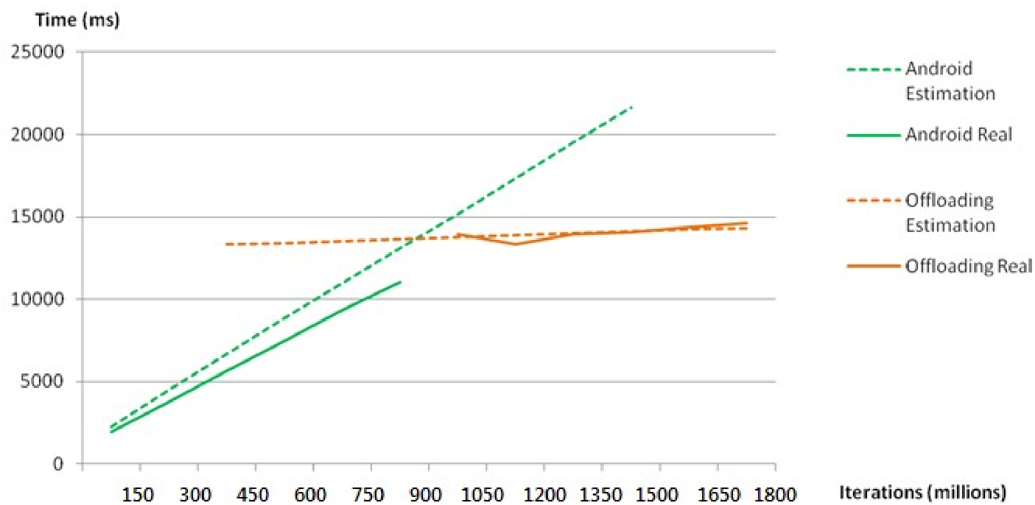

Figure 7: Behavior of the system with different amounts of computation, with 1MB of input data of the task.

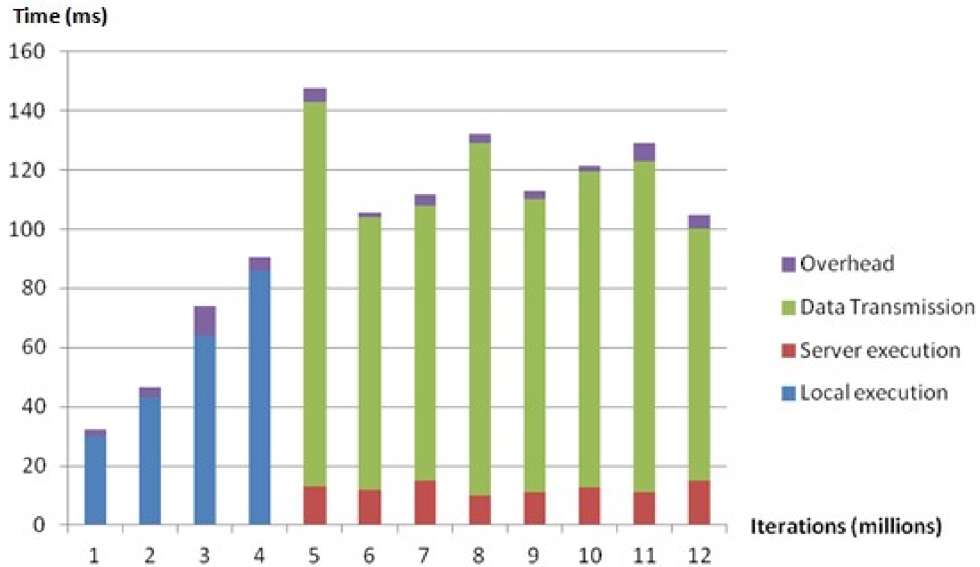

Figure 8: Breakdown of many executions of a simple looping algorithm managed by the proposed MCO system. In the X axis, the millions of iterations that the algorithm have looped. In the Y axis, the execution time that it has taken.

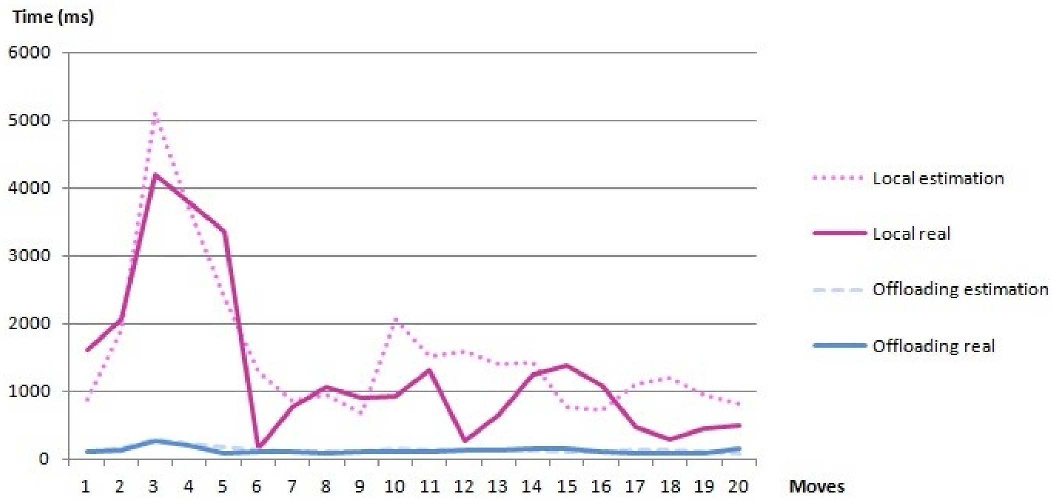

Figure 9: Executions of the AI (in a hard difficulty level) of a chess game managed by the proposed MCO system.

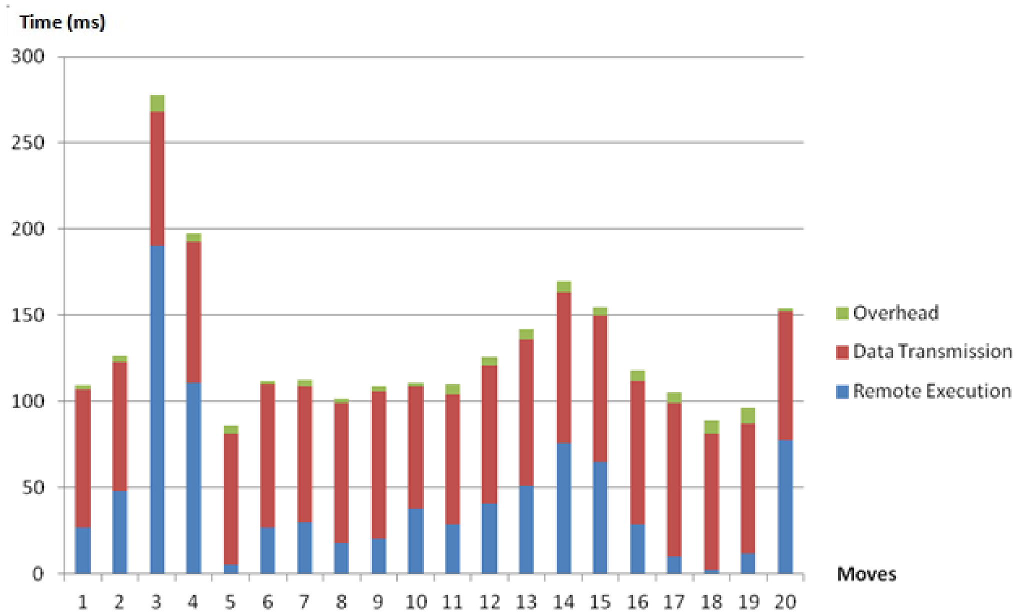

Figure 10: Breakdown of many executions of the AI (in a hard difficulty level) of a chess game managed by the proposed MCO system. In the X axis, the moves ordered as the game evolved. In the Y axis, the execution times.

3.4.3 Interpretation

In the figures from 4 to 7 it can be observed that the estimations produced by the system are quite accurate, thanks to the feature of forecasting the data transferring times taking into account both the bandwidth and RTT. For applications sending an almost negligible amount of data, like a chess game or like the case shown in Figure 4, only the RTT matters. But as the data size grows, the RTT loses relevancy and the bandwidth plays the main role.

Although the figures are produced after averaging many tests, the randomness of the network is still noticeable. In figures 6 and 7 it can be observed that the real offloading times are not following a linear growth.

It is also of interest to observe the point where offloading starts being worth it. In Figure 4, only about 4 millions of iterations are needed to reach this point. But in the next 3 figures, the point arrives around 10, 170 and 850 millions, respectively. When a task needs a large input, offloading makes sense only if the amount of computation that it is going to perform is as well really large.

Note that in Figure 6 and Figure 7, these limit points correspond to approximately 3.75 seconds and 14 seconds of execution, respectively. Only the really computationally intensive applications will benefit from offloading if they need a big input. This could be the case of a face recognition application, that requires an image file as input -which may be big- and then uses many complex detection algorithms on it.

The MCO system proposed in this thesis can adapt to tasks of this nature, as well as it can adapt to tasks that require only an almost negligible input. These are the cases shown in the figures 8, 9 and 10.

In Figure 8, the prediction power of AESTET is tested with the same simple looping algorithm used in Figure 4, with a negligible input. The figure shows the breakdown of the real execution times. After 4 millions of iterations, the task is not executed locally anymore, managed by the predictions of AESTET. This option is correct, as it can be seen in Figure 4. The most interesting of Figure 8 are the overheads produced by the estimations of AESTET. It can be observed that the overhead of each execution is always between 1 and 10 milliseconds, which is acceptable.

In figures 9 and 10, AESTET starts with an initial database of 20 samples. In the

case of the AI of the chess game, the input representation is defined as a number that identifies uniquely each input, and at the same time, the inputs with similar difficulty levels have similar input representation values.

Thus, when managing the task corresponding to the AI of the chess game shown in Figure 9, the MCO system will always decide that it is beneficial to offload it, as the difficulty level is hard, and the previously generated database indicates huge execution times for this difficulty level. Figure 10 corresponds to the breakdown of the real executions (all offloaded) of Figure 9. Figure 10 shows that the overhead produced by the estimations of AESTET is also very reduced. In [Mar13] other easier difficulty levels of the chess game with lesser amounts of computation are tested.

Note that for the generation of the figures 9 and 10, the initial database was reseted for each new execution, as it was important to start from the same state (when an execution is done, new entries are added to the database, and the prediction power of AESTET grows, so with no reset the starting state would have changed).

When using a not reseted database, AESTET is generally able to predict very well the first moves of the game, as these are similar in all the games and have been done many times before. As the game evolves, situations of the board that never occurred before are approached, and the predictions of AESTET are less accurate. However, even for one of these cases, AESTET will make a good prediction, as it will find in the database k-NN with the same difficulty level. The only critical point would be a difficulty level with random execution times, for which sometimes it would be worth it to offload and sometimes not. Fortunately, this is not the case (for more details refer to [Mar13]).

3.5 Further work

In the first place, it was emphasized that the MCO system presented in this paper includes many simplifications, as many features of the real MCO systems were not needed for the purposes of the thesis. The system could be extended adding the missing features: automatic partitioning, virtualization at a better level than a JVM, serialization, multiple users and a more complex surrogates' infrastructure. This would notably increase the usability and compatibility of the system.

The usability of the implementation of the system for the client side could also be

improved using the programming technique of reflection. Right now the software is distributed with the original Java source files, and the developer must make a few modifications on one of them. Reflection could avoid this, and then the system could be distributed as a compiled JAR library.

At the moment, the system does not allow the possibility to maintain an state between the surrogate and the client. Adding this feature would be useful for application like real time games.

The design of AESTET is acceptable but its implementation could be optimized. The way in which the database is queried to obtain the k-NN described in section 3.2.3 is not optimal, and the later processing of the obtained set of entries can as well be improved. However, as the system works only with input representations (single numeric values), it is already quite efficient as is, and it has been shown that the overhead produced is minimal.

4 Conclusions

As a general conclusion, the MCO system presented in this thesis is able to improve the performance of mobile devices' applications but requires their developers to do many adaptations, since in comparison with real MCO systems, it has many simplifications. It has been shown that in most of the cases, improving the performance can be translated into saving energy, especially when the input parameters of the offloadable task have a small size.

In order to take an offloading decision, the MCO system needs to forecast both the execution time of a task and the network state. For the first, the prediction mechanism AESTET is used. For the later, the system simply obtains the properties of the network at the beginning of the execution of an application, and assumes them to be similar during the whole execution of the application, if not changing to another network. This has been proved to be an acceptable approach for the purposes of this thesis.

Although not being very accurate, the results show that the estimations of AESTET are precise enough to take the correct offloading decision in most of the cases. AESTET follows a scheme originally designed for estimating the execution times of large tasks (e.g. for grid computing), where the overhead produced by computing the estimations is negligible. It was a challenge to see if this scheme could fit in the context of MCO. It was realized that estimating the execution times of complex tasks with multiple input parameters produced too much overhead, following the original scheme. Considering this, the scheme was redesigned so that the inputs would have to be translated into a single numeric representation. This would have to be done by the applications' developers, decreasing the usability of the system. Furthermore, the translation functions might be difficult to define in some cases, and there might even be tasks with complex inputs untranslatable to a single numeric representation. The accuracy of AESTET depends on the reliability of these translation functions.

The MCO system proposed in this thesis can be useful for any real application suitable for MCO. However, the system is currently only implemented for Android, and the virtualization is limited to a JVM. More open source Android applications with pure Java computationally intensive tasks were expected to be found .

Most of the tasks suitable for MCO have significantly variable execution times

depending on their inputs, and AESTET takes advantage of this to make the predictions. If the scope of the MCO system would only comprise the applications with significantly variable execution times depending on the size of the input, then AESTET could gain a lot of usability. The developer would no longer need to provide to the system translation functions capable to summarize the input of his tasks into a single numeric representations. Furthermore, in this case it could be a better approach that the applications using the system would follow a training phase to generate the initial samples database needed for AESTET. Going further, automatic partitioning of the applications could be considered then, as no developer-provided information would be needed for the tasks. Finally, the identified tasks could be automatically uploaded from the mobile devices to the surrogates in case of not being already there, instead of requiring the developer to do it beforehand. If all of these changes were to be carried out, the developer interaction would be reduced to none, and the tools of the web interface would not be needed anymore. The usability of the MCO system would hence be radically improved.

Taking into account the different exposed points, it can be affirmed that the system can be used as is for future experimentation, and thereby fulfills the parallel aim of this thesis.

References

[ADT13] ADT Eclipse plugin. http://developer.android.com/sdk/eclipse-adt.html, May 2013.

[AID13] AIDL. http://developer.android.com/guide/topics/fundamentals.html, May 2013.

[AMa13] Android Market. http://www.android.com/market/, April 2013.

[And13] Android Developers. http://developer.android.com, April 2013.

[AOS04] X. Gu *"Adaptive Offloading for Pervasive Computing"*, *IEEE Pervasive Comp.*, vol. 3, no. 3, pp.66 -73 2004.

[APS13] Android Play Store https://play.google.com/, April 2013.

[ATT13] AT&T Labs Research - Leading Invention, Driving Innovation: http://www.research.att.com/articles/featured_stories/2011_03/201102_Energy_efficient, June 2013.

[AWS13] Amazon Elastic Computing. http://aws.amazon.com/ec2/, May 2013.

[Bal03] R. K. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. *Tactics-Based Remote Execution for Mobile Computing.* In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys), San Francisco, CA, 2003.

[Bar13] Barbera, M.V.; Kosta, S.; Mei, A.; Stefa, J., *"To offload or not to offload? The bandwidth and energy costs of mobile cloud computing"*, *INFOCOM, 2013 Proceedings IEEE* , vol., no., pp.1285,1293, 14-19 April 2013.

[CAi11] C. Ai, J. Liu, C. Fan, X. Zhang, and J. Zou, *"Enhancing personal information security on android with a new synchronization scheme*", in Proc. of WiCOM 2011, 2011.

[CDA11] Kemppainen, M. (2011). *Mobile computation offloading: A context-driven approach.* Aalto University. T-110.5190 Seminar on Internetworking.

[Chi93] Philip F. Chimento and K. S . Trivedi. *The Completion Time of Programs on Processors Subject to Failure and Repair.* IEEE Transactions on computers, Vol. 42, No. 10, October 1993.

[Chu10] B.-G. Chun and P. Maniatis. *Dynamically partitioning applications between weak devices and clouds.* In Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, MCS '10, pages 7:1–7:5, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0155-8.

[Chu11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik and Ashwin Patti. *CloneCloud: Elastic Execution between Mobile Device and Cloud.* In proceedings of the sixth conference on Computer systems, April 10–13, 2011, Salzburg, Austria.

[Clo09] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, *"The Case for VM-Based Cloudlets in Mobile Computing"*, IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct. 2009.

[Clo12] S. Simanta, K. Ha, G. Lewis, E. Morris, and M. Satyanarayanan, *"A Reference Architecture for Mobile Code Offload in Hostile Environments"*, in Fourth International Conference on Mobile Computing, Applications and Services, Seattle, WA, October 2012.

[Clo13] Class Cloudlet.
http://www.cloudbus.org/cloudsim/doc/api/org/cloudbus/cloudsim/Cloudlet.html, May 2013.

[CMU13] CMU Sphinx. http://cmusphinx.sourceforge.net/, May 2013.

[Cpl13] Cplusplus. http://www.cplusplus.com/, May 2013.

[CSR13] Cpu Spy Reborn. http://mirko-ddd.xda-developers.com/cpu-spy-reborn, June 2013.

[Cue10] Cuervo, E. et al. (2010). *MAUI: making smartphones last longer with code offload.* In Proceedings of the 8th international conference on Mobile systems, applications and services, San Francisco, CA.

[Din11] Hoang T. Dinh, Chonho Lee, Dusit Niyato and Ping Wang. *"A survey of mobile cloud computing: architecture, applications, and approaches"*, School of Computer Engineering, Nanyang Technological University (NTU), Singapore, Published online in Wiley Online Library, 2011.

[Dou91] F. Douglis and J. Ousterhout. *Transparent Process Migration: Design Alternatives and the Sprite Implementation.* Software - Practice and Experience, 21(8):757–785, August 1991.

[Ecl13] Eclipse. http://www.eclipse.org/, April 2013.

[Erc13] *Automatic Offloading of Mobile Applications Using Evolutionary Algorithms.* In ERCIM NEWS online edition. http://ercim-news.ercim.eu/en93/special/automatic-offloading-of-mobile-applications-using-evolutionary-algorithms, May 2013.

[Euc13] Eucalyptus. http://www.eucalyptus.com/, June 2013.

[Eye13] eyes-free Speech Enabled Eyes-Free Android Applications.
http://code.google.com/p/eyes-free/, June 2013.

[FGo13] Free Go Programs. http://www.gnu.org/software/gnugo/free_go_software.html, June 2013.

[Fli02] J. Flinn, S. Park, and M. Satyanarayanan, *"Balancing Performance, Energy, and Quality in Pervasive Computing"*, Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS 02), IEEE CS Press, 2002, pp. 217–226.

[Flo13] H. Flores, S. N. *Srirama: Adaptive Code Offloading and Resource-intensive Task Delegation for Mobile Cloud Applications*, The 11th International Conference on Mobile Systems, Applications and Services (MobiSys 2013), June 25-28, 2013. ACM.

[FUB13] The proposed offloading system. https://www.mi.fu-berlin.de/offload/, April 2013.

[Gao12] B. Gao, L. He, L. Liu, K. Li, and S.A. Jarvis, *"From Mobiles to Clouds:*

Developing Energy-Aware Offloading Strategies for Workflows", in Proc. GRID, 2012, pp.139-146.

[Giu09] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. *Calling the cloud: Enabling mobile phones as interfaces to cloud applications*. In Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, pages 1–20, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[GM13] Griera, M. and Martínez, J. *"Mobile devices computation offloading"*, Softwareprojekt Mobilkommunikation, Institute of Computer Science, Department of Mathematics and Computer Science, Freie Universität Berlin, 2013.

[Gog13] Google Goggles. http://www.google.com/mobile/goggles/, April 2013.

[GoL13] Tesuji Software Go Library. http://sourceforge.net/projects/tesujigolibrary/, June 2013.

[Goo13] Google Scholar. http://scholar.google.com/, May 2013.

[IEE13] IEEE Xplore Digital Library. http://ieeexplore.ieee.org/, May 2013.

[Ini13] Inimesed. http://kaljurand.github.io/Inimesed/, April 2013.

[IoT13] The Internet of Things. http://standards.ieee.org/innovate/iot/, April 2013.

[Ive96] M. Iverson, F. Ozguner and G. Follen *"Run-Time Statistical Estimation of Task Execution Times for Heterogeneous Distributed Computing", Proc. High Performance Distributed Computing Conf.*, pp.263 -270 1996.

[Jav13a] Java OCR. http://sourceforge.net/projects/javaocr/ , May 2013.

[Jav13b] Java Speech API. http://jsapi.sourceforge.net/, May 2013.

[Jav13c] Javabeat. http://www.javabeat.net/2007/04/the-java-6-0-compiler-api/, May 2013.

[Jun12] Juntunen, A.; Kemppainen, M.; and Luukkainen, S. *"MOBILE COMPUTATION OFFLOADING - FACTORS AFFECTING TECHNOLOGY EVOLUTION"*. 2012 International Conference on Mobile Business. Paper 9.

[Kem10] Kemp, R., Palmer, N., Kielmann, T. and Bal, H. (2010). *Cuckoo: a Computation Offloading Framework for Smartphones.* In Proceedings of The Second International Conference on Mobile Computing, Applications and Services, Santa Clara, CA.

[Kov12] D. Kovachev, Tian Yu, R. Klamma, *"Adaptive Computation Offloading from Mobile Devices into the Cloud"*, in proc. of the IEEE 10th intl. Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 784-791, Jul. 2012.

[Kri10] Kristensen and Niels Olof Bouvin. 2010. Scheduling and development support in the Scavenger cyber foraging system. *Pervasive Mob. Comput.* 6, 6 (December 2010), 677-692.

[Kul87] V. G. Kulkarni; V. F. Nicola; K. S. Trivedi . *The Completion Time of a Job on Multimode Systems.* Advances in Applied Probability, Vol. 19, No. 4. (Dec., 1987), pp. 932-954.

[Kum10] K. Kumar and Y. Lu, *"Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?"*, IEEE Computer, vol. 43, no. 4, pp. 51-56, 2010.

[Kum12] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava, *"A Survey of Computation Offloading for Mobile Systems"*, Mobile Networks and Applications, April 2012.

[Lag11] Lagerspetz, E.; Tarkoma, S., "Mobile search and the cloud: The benefits of offloading," *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on* , vol., no., pp.117,122, 21-25 March 2011.

[Lee13] Kyunghan Lee; Joohyun Lee; Yung Yi; Injong Rhee; Song Chong, "Mobile Data Offloading: How Much Can WiFi Deliver?," *Networking, IEEE/ACM Transactions*, vol.21, no.2, pp.536,550, April 2013.

[Mar13] Martínez, J. *"Improving the performance and usability of an offloading engine for Android mobile devices with application to a chess game"*, Master's Thesis, Institute of Computer Science, Department of Mathematics and Computer Science, Freie Universität Berlin, 2013.

[MMP95] Y A Li, J K Antonio, H J Siegel, M Tan and D K Watson. "Estimating the Distribution of Execution Times for SIMD/SPMD Mixed-Mode Programs", In Proc. of the Heterogeneous Computing Workshop, 1995.

[Msg13] Message Pack. http://msgpack.org/, June 2013.

[Mul90] S. Mullender, G. van Rossum, A. Tanenbaum, R. van Renesse, and H. van Staveren. *Amoeba - A Distributed Operating System for the 1990s*. IEEE Computer, 23:44–53, 1990.

[New09] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden. *Wishbone: Profile-based Partitioning for Sensornet Applications*. In Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI), Boston, MA, April 2009.

[Nob97] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. *Agile Application-Aware Adaptation for Mobility*. In Proc. of the ACM Symposium on Operating System Principles (SOSP), 1997.

[OCR13a] Android OCR. https://github.com/rmtheis/android-ocr, June 2013.

[OCR13b] Online OCR API. http://ocrapiservice.com/documentation/, May 2013.

[Ora13] Oracle Documentation. http://docs.oracle.com/, May 2013.

[PAN13a] PANDA Software - User Manual.
http://privatewww.essex.ac.uk/~kunyang/Projects/EPSRC/PANDA-software.html, May 2013.

[PAN13b] Policy-based Model-driven Pervasive Service Creation and Adaptation: PANDA.
http://privatewww.essex.ac.uk/~kunyang/Projects/EPSRC/PANDA.html, May 2013.

[Par11] JiSu Park; HeonChang Yu; KwangSik Chung; Eunyoung Lee, "Markov Chain Based Monitoring Service for Fault Tolerance in Mobile Cloud Computing," *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on* , vol., no., pp.520,525, 22-25 March 2011. doi: 10.1109/WAINA.2011.10

[PCA13] Pocket Chess for Android. http://code.google.com/p/pocket-chess-for-android/, April 2013.

[PMo13] Mobile Device Power Monitor. http://www.msoon.com/LabEquipment/PowerMonitor/, May 2013.

[PTu13] Power Tutor. http://powertutor.org/, May 2013.

[Rei13] Reign Design Blog. http://www.reigndesign.com/blog/using-your-own-sqlite-database-in-android-applications/, May 2013.

[Rei94] Brian Reistad, David K. Gifford. *Static dependent costs for estimating execution time*. ACM SIGPLAN Lisp Pointers, v.VII n.3, p.65-78, July-Sept. 1994.

[Sat96] M. Satyanarayanan "Fundamental Challenges in Mobile Computing", *Proc. ACM Symp. Principles of Distributed Computing*, pp.1 -7 1996.

[Sha13] Shazam. http://www.shazam.com, May 2013.

[Son13] Sony Add-on SDK. http://developer.sonymobile.com/knowledge-base/sony-add-on-sdk/install-the-sony-add-on-sdk/, May 2013.

[Sph13] Sphinx-4 speech recognizer. http://cmusphinx.sourceforge.net/sphinx4/, May 2013.

[SQL13] SQ Lite. http://www.sqlite.org/, April 2013.

[SQL13] SQLite JDBC library. http://code.google.com/p/sqlite-jdbc/, May 2013.

[Sql13] sqlite-jdbc. https://bitbucket.org/xerial/sqlite-jdbc, September 2013.

[Sta13] Stackoverflow. http://stackoverflow.com, April 2013.

[Sti10] V. Stirbu. *A RESTful architecture for adaptive and multi-device application sharing.* In Proceedings of the First International Workshop on RESTful Design, WS-REST '10, pages 62–65, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-959-6.

[Tes13] Tesseract OCR engine. https://code.google.com/p/tesseract-ocr/, May 2013.

[Til06] Tilevich E, Smaragdakis Y (2006) *J-orchestra: automatic Java application partitioning*. In: European conference on object- oriented programming, pp 1–3.

[Tom13] Apache Tomcat. http://tomcat.apache.org/, May 2013.

[TTS13] FreeTTS 1.2 speech synthesizer. http://freetts.sourceforge.net/, May 2013.

[Wan13] Wang, Q., Griera, M., Martínez, J. and Wolter, K. *"Analysis of Local Re-execution in Mobile Offloading System"*, Institute of Computer Science, Department of Mathematics and Computer Science, Freie Universität Berlin, 2013.

[Wei08] Y. Weinsberg, D. Dolev, T. Anker, M. Ben-Yehuda, and P. Wyckoff. *Tapping into the Fountain of CPUs – On Operating System Support for Programmable Devices*. In Proc. of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2008.

[Wol08] R. Wolski, S. Gurun, R. Krintz, and D. Nurmi, *"Using bandwidth data to make computation offloading decisions"*, in in Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2008), High-Performance Grid Computing Workshop, 2008.

[Xda13] XDA Developers. http://forum.xda-developers.com/, May 2013.

[XGu03] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. *Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments*. In Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom), 2003.

[XGu04] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt, *"Adaptive Offloading for Pervasive Computing",* IEEE Pervasive Computing, vol. 3, pp. 66–73, July 2004.

[Xia07] Xian C, Lu Y-H, Li Z (2007) *Adaptive computation offloading for energy conservation on battery-powered systems*. In: International conference on parallel and distributed systems, pp 1–8.

[Yan08] Kun Yang and Shumao Ou (University of Essex), Hsiao-Hwa Chen (National Sun Yat-Sen University). *"On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications"*. IEEE Communications Magazine, pp.53-63, January 2008.

[Yoo12] C. Yoon, D. Kim, W. Jung, and C. Kang, *"Appscope: Application energy metering framework for android smartphone using kernel activity monitoring,"* in Proc. of USENIX ATC 12, 2012.

[You01] C. Young and Y. N. Lakshman. *Protium, an Infrastructure for Partitioned Applications*. In Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Schloss Elmau, Germany, May 2001.

[YSu05] Y.-Y. Su and J. Flinn. *Slingshot: Deploying Stateful Services in Wireless Hotspots*. In Proc. of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys), Seattle, WA, June 2005.

[Zha10] Xinwen Zhang, Sangoh Jeong, Simon Gibbs, and Anugeetha Kunjithapatham. *Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms*. In the 3rd International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications (MobilWare), 2010.

[ZLi01] Li Z, Wang C, Xu R (2001) *Computation offloading to save energy on handheld devices: a partition scheme*. In: International conference on compilers, architecture, and synthesis for embedded systems, pp 238–246 .