TABLE OF CONTENTS

Page

INTRODUCTION 1
0.1 Motivations 1
0.2 Problem Statement 3
0.3 Main Goal 4
0.4 Methodology 5
0.5 Technical Contributions 11
0.6 Publications 12
0.7 Thesis Organization 13

LITERATURE REVIEW 15
1.1 Mobile Virtualization 15
1.2 Mobile Computation Offloading 16
1.3 Predictive Management Techniques for Virtual Environments 19
1.4 Dynamic Offloading Algorithms 20
1.5 Conclusion 22

CHAPTER 2 ARTICLE 1: SELECTIVE MOBILE CLOUD OFFLOADING TO AUGMENT MULTI-PERSONA PERFORMANCE AND VIABILITY 23
2.1 Abstract 23
2.2 Introduction 24
2.3 Background and Related Work 27
2.3.1 Mobile Virtualization 27
2.3.2 Offloading 29
2.3.3 Proposed Approach Positioning 32
2.4 Problem Illustration 33
2.5 Offloading Meets Multi-Persona 35
2.6 Multi-Objective Optimization for Multi-Persona 37
2.6.1 Problem Definition 37
2.6.2 Problem Formulation 40
2.7 Heuristic Algorithms for Optimal Distribution of Multi-Persona Components 42
2.7.1 Representation of Individuals 43
2.7.2 Fitness Evaluation 43
2.7.3 Operators 43
2.7.4 Algorithm and Time Complexity Analysis 44
2.8 Implementation and Experiments 45
2.8.1 Implementation 45
2.8.2 Experiments 47
2.8.2.1 Testbed Setup 48

2.8.2.2 Assumptions 50
2.8.2.3 Results and Analysis 50
2.9 Conclusion and Future Directions 56

CHAPTER 3 ARTICLE 2: SMART MOBILE COMPUTATION OFFLOADING: CENTRALIZED SELECTIVE AND MULTI-OBJECTIVE APPROACH 59
3.1 Abstract 59
3.2 Introduction 60
3.3 Computations Offloading Overview 63
3.4 Related Work 65
3.5 Technical Problems 69
3.5.1 Accuracy and Overhead of Decision Model Evaluation 69
3.5.2 Decision Model Metrics 70
3.6 Centralized Selective and Multi-Objective Offloading: Insights 71
3.7 Selective Mechanism 73
3.7.1 Hotspots Profiling 73
3.7.2 Hotspots Detection 74
3.7.3 Selection Algorithm 74
3.8 Centralized Selective Offloading Decision Model 76
3.8.1 Definition 77
3.8.2 Model Formulation 78
3.9 Intelligent Decision Making Process 80
3.9.1 Solution Encoding 82
3.9.2 Fitness Evaluation 82
3.9.3 Evolution Process 83
3.9.3.1 Selection 83
3.9.3.2 Crossover 83
3.9.3.3 Mutation 84
3.10 Numerical Analysis 84
3.10.1 Testbed Setup 84
3.10.2 Results 86
3.10.2.1 Decision Model Efficiency 86
3.10.2.2 Selective Mechanism and Intelligent Decision Maker Efficiency 87
3.11 Conclusion and Future Directions 91

CHAPTER 4 ARTICLE 3: COST-EFFECTIVE CLOUD-BASED SOLUTION FOR MULTI-PERSONA MOBILE COMPUTING IN WORKPLACE 93
4.1 Abstract 93
4.2 Introduction 94
4.3 Computation offloading to support mobile devices: Background 97
4.4 Related Work 99

4.4.1 Mobile Centric Offloading 100
4.4.2 Cloud Centric Offloading 102
4.4.3 Analysis 103
4.5 Illustrative Business Model and Problem Description 104
4.6 Cost-Effective Offloading: System Model 107
4.7 Collective Multi-Persona Offloading Optimization Problem (CMPO) 109
4.7.1 Definition 109
4.7.2 Formulation 112
4.8 Smart Cost-Effective Decision Maker 115
4.8.1 Solution Encoding 115
4.8.2 Fitness Evaluation 116
4.8.3 Evolution Process 117
4.8.3.1 Selection 117
4.8.3.2 Crossover 117
4.8.3.3 Mutation 118
4.9 Numerical Analysis 118
4.9.1 Setup 120
4.9.2 Generated distribution cost 122
4.9.3 Decision maker overhead 123
4.9.4 Satisfaction rate 124
4.9.5 Optimized decision maker speedup 126
4.9.6 Summary 126
4.10 Conclusion and Future Directions 127

CHAPTER 5 ARTICLE 4: PROACTIVE SOLUTION AND ADVANCED MANAGEABILITY OF MULTI-PERSONA MOBILE COMPUTING 129
5.1 Abstract 129
5.2 Introduction 130
5.3 Related Works 134
5.3.1 Predictive Virtual Instances Management Strategies 134
5.3.2 Dynamic Offloading Algorithms 135
5.3.3 Our Contributions 136
5.4 System Model 137
5.5 Machine Learning Prediction 140
5.5.1 Linear Regression 141
5.5.2 Support Vector Regression 141
5.5.3 Neural Network 142
5.5.4 Deep Neural Network 142
5.6 Problem Formulation 142
5.7 Proposed Dynamic Programming Algorithm 145
5.7.1 DP Table Filling 145
5.8 Evaluation 148

5.8.1 Setup 148
5.8.2 Numerical Analysis 150
5.9 Conclusion and Future Directions 158

CONCLUSION AND RECOMMENDATIONS 161

BIBLIOGRAPHY 165

LIST OF TABLES

Page

Table 2.1 Taxonomy of mobile code offloading approaches. 29

Table 2.2 Applications in each persona. 33

Table 2.3 Formulas notations. 40

Table 2.4 Distribution of services in different scenarios. 48

Table 2.5 Parameters 49

Table 2.6 Number of iterations. 49

Table 2.7 Distribution of services based on the decision making algorithm. 52

Table 2.8 Required number of iterations. 55

Table 3.1 Classification of offloading approaches. 65

Table 3.2 Decision error rate. 90

Table 4.1 Taxonomy of offloading schemes. 99

Table 4.2 Taxonomy of offloading schemes. 100

Table 5.1 Prioritization scheme. 139

Table 5.2 DP Table Filling 146

Table 5.3 Device Usage Behavior 149

Table 5.4 ML techniques setup for phase #1 149

Table 5.5 ML techniques setup for phase #2 149

Table 5.6 Usage scenarios. 156

Table 5.7 Generated strategies. 156

LIST OF FIGURES

Page

Figure 0.1 Mobile cloud offloading approach: objectives, contributions and architecture. 7

Figure 0.2 Selective approach: objectives, contributions and architecture. 8

Figure 0.3 Cost-effective approach: objectives, contributions and architecture. 9

Figure 0.4 Proactive advanced approach: objectives, contributions and architecture. 11

Figure 2.1 Multi-Persona efficiency and viability. 34

Figure 2.2 Proposed architecture. 35

Figure 2.3 Algorithms overhead on the mobile device. 51

Figure 2.4 Approach evaluation. 54

Figure 2.5 Optimal(OS) and Good(GS) solutions overheads. 56

Figure 3.1 Mobile code offloading architecture. 64

Figure 3.2 System model. 71

Figure 3.3 Decision savings. 87

Figure 3.4 Decision maker overhead. 88

Figure 3.5 Components overhead. 89

Figure 3.6 Overall overhead. 90

Figure 4.1 Mobile computation offloading. 98

Figure 4.2 Illustrative business model. 105

Figure 4.3 Unbalanced resource usage, performance, and monetary fees. 107

Figure 4.4 System model. 108

Figure 4.5 Encoding scheme. 115

Figure 4.6 Evolution-based crossover. 117

Figure 4.7 Mutation. 118

Figure 4.8 Generated solution cost: average local CPU usage, memory consumption, execution time, energy loss and monetary fees. 121

Figure 4.9 Decision maker overhead. 124

Figure 4.10 Satisfaction rate. 125

Figure 4.11 Optimized decision maker speedup. 126

Figure 5.1 System model. 138

Figure 5.2 DS1: observed CPU vs. predicted. 151

Figure 5.3 DS1: observed memory vs. predicted. 152

Figure 5.4 DS2: observed CPU vs. predicted. 153

Figure 5.5 DS2: observed memory vs. predicted. 153

Figure 5.6 DS3: observed vs. predicted values. 154

Figure 5.7 DS4:Observed vs. Predicted values. 154

Figure 5.8 DS5:Observed vs. Predicted values. 155

Figure 5.9 DS6:Observed vs. Predicted values. 155

Figure 5.10 Strategies efficiency. 157

Figure 5.11 Decision engine performance. 158

LIST OF ABREVIATIONS

ANN/NN	Artificial Neural Network
BYOD	Bring Your Own Device
DNN	Deep Neural Network
DP	Dynamic Programming
GA	Genetic Algorithm
KNN	K-Nearest Neighbor
LR	Linear Regression
ML	Machine Learning
MCC	Mobile Cloud Computing
RF	Random Forest
SVM	Support Vector Machines
SVR	Support Vector Regression
VM	Virtual Machine
VP	Virtual Phone

INTRODUCTION

0.1 Motivations

Technology has transformed even the smallest tasks in our daily life. As the whole world is going into the new phase of technological advances, our needs become more sophisticated. We are looking for speed, quality, mobility and effectiveness and on the other hand, we need these features to be integrated in a solution small enough to be carried in pocket. With the growing speed of technological advancements, smartphones have become the essential components of our everyday performance. Equipped with advanced operating systems, smartphones have risen in prominence and consumer habits have shifted, relegating desktop to mobile computing.

In this mobile-first world, business environment has accordingly entered a new era of mobile-led changes, which have revolutionised the trend of bring your own device (BYOD). This policy has been around for a while and is gaining momentum as more and more businesses are looking to improve work efficiency and decrease operational costs. Simply, "Bring Your Own Device" refers to employees having the ability to opt their personally owned mobile devices to perform business tasks. However, letting staff use their own gadgets could rapidly turn into a headache for IT. Potential pitfalls that a company may face because of BYOD are all around security and confidentiality, with personal devices accessing corporate network and storing sensitive business data. Such key considerations force IT to typically apply some security policies on end devices, which in turn raise some privacy issues for the end users who might sacrifice their personal data to comply with such policies. The BYOD debate has received much coverage in recent years [Rouse (2012a)]. Mobile devices with dual persona were released to determine this battle, enabling two phones-in-a-phone, typically, one for private personal use and another for business use. This technology provides a way to keep corporate applications and their associated data segregated and protected on an employee's personal mobile device in

a separate and independent end user environment, while IT can only see and manage the assets in the business environment, maintaining end user's privacy.

However, nowadays, multi-persona has become a game changer for mobile devices [Eiferman (2014a)]. Multi-persona creates the same impenetrable wall between an employee's applications and an organization's data and applications, yet allowing one phone to co-host not only two, but more completely independent and secure virtual environments. But why would anybody want so many personas (virtual environments)? A user can isolate private banking services and e-commerce, sensitive corporate data, social networking and games in separate personas. Such isolation provides an efficient management of financial transactions, prevents untrusted applications from accessing critical information and allows sharing the device with other family members without ending up with accidental phone calls, unintended in-app purchases or even access to restricted content. Even more interesting use cases come with having multiple work personas with different levels of security. For example, while working at their private clinic and at multiple hospitals, doctors are subject to different mobile policies, reflecting each of the different institutions. With personal, clinic and hospitals personas, multi-persona allow doctors to comply with the policy of each and effectively treat their patients while maintaining their own unburdened personal use of the device. Whether in these or any other example, the success of multi-persona lies in its capability of consolidating multiple mobile devices on a single terminal, while making the latter able to clearly distinguish between the different contexts in which it is used.

With the option to tap into virtualization solution, a user can differentiate between these contexts through virtual phone dedicated for each. Mobile virtualization is one of the key technologies behind multi-persona realization. Similar to virtualization on servers and desktop machines, mobile virtualization allows multiple virtual environments (instances) to run simultaneously on the same physical mobile device. These instances are called personas or virtual

phones (VPs). A significant added value of virtualization is the ability to isolate the virtual instances from each others so failures in one area do not affect other areas. Yet, to realize multi-persona, mobile virtualization is much more challenging since it requires a trade-off between secure isolation and scalability of personas on mobile devices having limited computation capabilities, memory capacity and battery lifetime. Different techniques can be used to implement mobile virtualization. *System-Level Virtualization* [Barr *et al.* (2010)] is a technique that offers the ability to run multiple operating systems on one physical device using an additional software layer called a hypervisor (or microkernel) [Wessel *et al.* (2013)]. Yet, this technique imposes high overhead due to the complete software stack in each virtual environment (i.e., Kernel, Middleware, Apps). Therefore, it might be suitable for dual-persona but not for multi-persona functionality. There is also *User-Level Isolation* [Android (2014)], in which both the kernel and the middleware layers are shared between the virtual instances. Yet this technique does not create virtual environments but rather wraps applications to separate them from each other, which does not realize the needed isolation for multi-persona. The last technique is *OS-Level Virtualization*, also called *Container-based virtualization*, which is a method that shares the kernel layer to run multiple virtual instances on a single operating system [Andrus *et al.* (2011)]. Allocating a minimum set of resources for each persona results in a collection of lightweight personas inside the device making the available OS resources enough for running more than two personas and thus the most adequate technique to realize multi-persona functionality.

0.2 Problem Statement

The growing wave of consumerization, expectation and sophistication of mobile devices has created a powerful force for change. With the significant development of mobile hardware such as multi-core CPUs, larger memory, network and high-resolution displays, today's mobile applications are becoming more complex with an increasingly feature-rich nature. However, no

matter how advanced mobile devices hardware is growing, it's still limited compared to its counterpart of desktop machines. The resource demands of mobile applications often outstrip the hardware capacities of mobile devices in terms of computations, memory and battery, and an additional virtualization layer just make things worse. Even with the lightweight virtualization techniques, co-hosting multiple virtual phones on a single resource constrained mobile device imposes high overhead on the latter leading to performance degradation and more critically to system crash due to lack of resources. Thorough experiments have been conducted in this work aiming to emphasize on this problem. Practically, we varied the number of personas running diversity of lightweight, moderate and heavy applications, and we examined the resource consumption on the device which affect personas lifetime, as well as the execution time to study their influence on the applications performance. With two personas, the results showed drastic increase in the CPU usage recording up to 62% compared to 26% with one persona along with significant increase in the energy consumption that reached 420 J compared to 329 J with one running persona. As for the execution time, it took up to 1380 s to finish the execution of the running applications, which is 1.7 times more than the case of running one persona. Another critical observation was also exposed in third scenario that involved 3 personas, where it was impossible to run the same applications as the personas kept shutting down due to lack of memory on the mobile terminal as well as the high consumption of other resources. These results reveal the incompetence of the mobile device resources to sustain high performing personas neither to tolerate their viability.

0.3 Main Goal

While personas would require to be scaled up, which is not an option as they are ported on a single resource constrained device, this thesis aims to propose a novel proactive mobile cloud computing solution with advanced manageability to boost the performance of the virtual instances and augment the viability of the physical host device.

0.4 Methodology

To emphasize the claimed technical problems, a virtualization architecture has been ported on a mobile device and different virtual environments were built. Extensive experiments have been performed throughout the work depicting the impediment of the limited computation capability, memory capacity and battery lifetime of the mobile terminal to support multi-persona without performance degradation or shorter-time system viability. Advanced technologies were investigated and novel techniques and algorithms were proposed in this thesis, aiming to reinforce multi-persona mobile computing.

Mobile Cloud Computing (MCC) is a new paradigm for mobile applications whereby the data processing and storage are moved from the mobile device to powerful and centralized computing platforms located in the cloud [Dinh *et al.* (2013)]. There are several existing definitions of mobile cloud computing, and different research alludes to different concepts of the 'mobile cloud' [Fernando *et al.* (2013)]. Commonly, the term mobile cloud computing means to run an application on a remote resource-rich server, while the mobile device acts like a thin client connecting over to the remote server through WiFi or cellular data network. Another approach is to consider other mobile devices themselves as resource providers of the cloud making up a mobile peer-to-peer network. Thus, the collective resources of the various mobile devices in the local vicinity, and other stationary devices too if available, will be utilized. This is called ad hoc approach, which presents several challenges. The first challenge is how to motivate surrogates to collaborate their resources? An interesting method is using common goals, but in the absence of common activities this will not prevail. In the case of monetary incentives, several questions need to be answered such as; how are credits represented in a mobile cloud? how will monetary transactions proceed in a secure method? how will the price of resources be decided? The second challenge is how to deal with malicious surrogates? The cloudlet concept is another approach to mobile cloud computing, where the mobile device offloads its workload

to a local 'cloudlet' comprised of several multi-core computers with connectivity to the remote cloud servers. These cloudlets would be situated in common areas such as coffee shops so that mobile devices can connect and function as a thin client to the cloudlet as opposed to a remote cloud server which would present latency and bandwidth issues. Although cloudlets can decrease latency, but they does not fully support a mobile user who needs to work while on the move. Cloudlets offload jobs to local resource rich server that could only support the needs of mobile device users who are within a limited range. Ad hoc cloud is not subject of this research work, which adopts the first cloud architecture. However, cloudlets could be used as complementary scheme.

A particularly popular technique to extend limited mobile hardware is computation offloading which migrates the execution of mobile functionalities to a powerful cloud-based server. A full research domain of computation offloading has been triggered within mobile cloud computing. Code offloading is an opportunistic process that leverages cloud resources (e.g., servers) to execute computations designated by a mobile terminal. Many computations offloading techniques [(Hung *et al.*, 2012; Cuervo *et al.*, 2010; Kosta *et al.*, 2012; Kemp, 2014; Chen *et al.*, 2012; Chun *et al.*, 2011; Shi *et al.*, 2014; Chae *et al.*, 2014; Gordon *et al.*, 2012; Flores *et al.*, 2014; Xia *et al.*, 2014)] have been proposed allowing mobile devices to migrate the execution and throw the burdens of computations to remote resourceful infrastructure. Offloading can be applied either on a full application or more fine-grained tasks like services, methods, or threads, releasing the mobile device from intensive processing. An offloading decision is taken based on a cost model that can estimate where the execution is more effective for the end device. The network bandwidth and latency, the resource demands of tasks and the mobile device state, all form a context that influences offloading efficiency. Due to mobility and the variation in all these aspects, the evaluation of this model changes from one execution to another and hence lead to different decisions. Tasks, applications and components are used exchangeably throughout the thesis.

While through these approaches, computation offloading has indeed proved its ability to enhance the performance and save energy on the mobile terminal, we started to build our solution based on these premises. Throughout this work, various research questions have arisen and hence several **objectives** had to be set, which are highlighted throughout the articles presented in the following chapters.

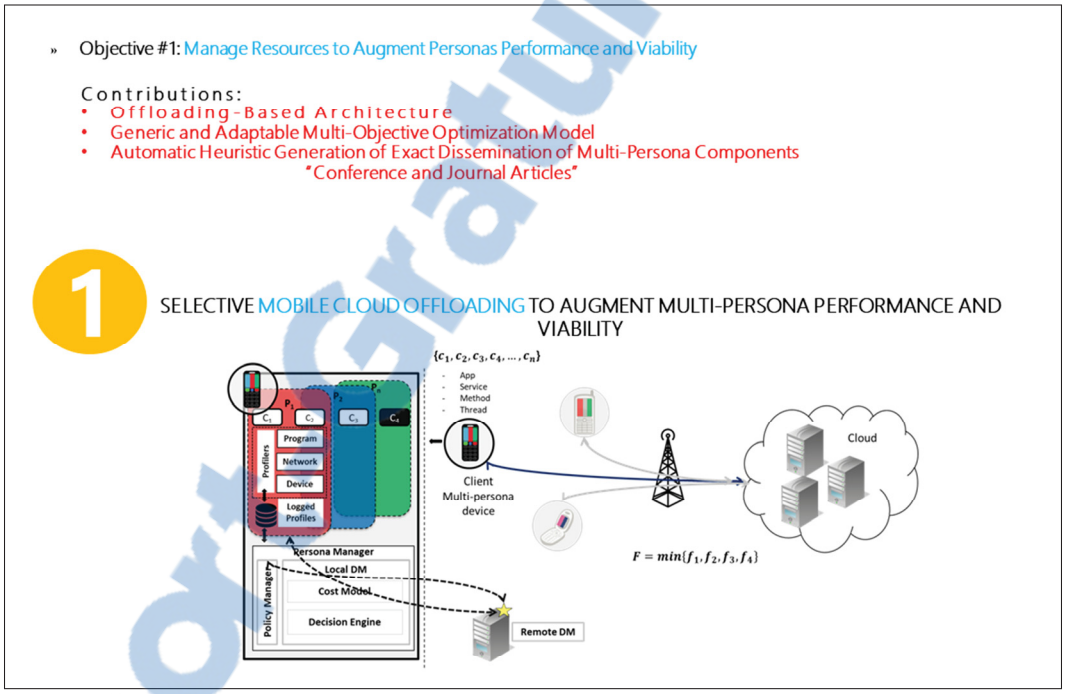


Figure 0.1 Mobile cloud offloading approach: objectives, contributions and architecture.

The aim of the first article (Chapter 2) is to efficiently manage the resources on the mobile terminal in order to boost personas performance and viability. In this regard, we introduce a mobile-cloud offloading approach as shown in Figure 0.1 with the following sub-objectives:

- Propose an offloading-based architecture to augment multi-persona performance and viability.

- Build a generic optimization model, for multi-persona, independent of the offloading granularity and adaptable to different execution contexts.

- Automatically generate exact distribution strategies (i.e., remote/local execution) of multi-persona applications to optimize both resource usage and performance on the physical device.

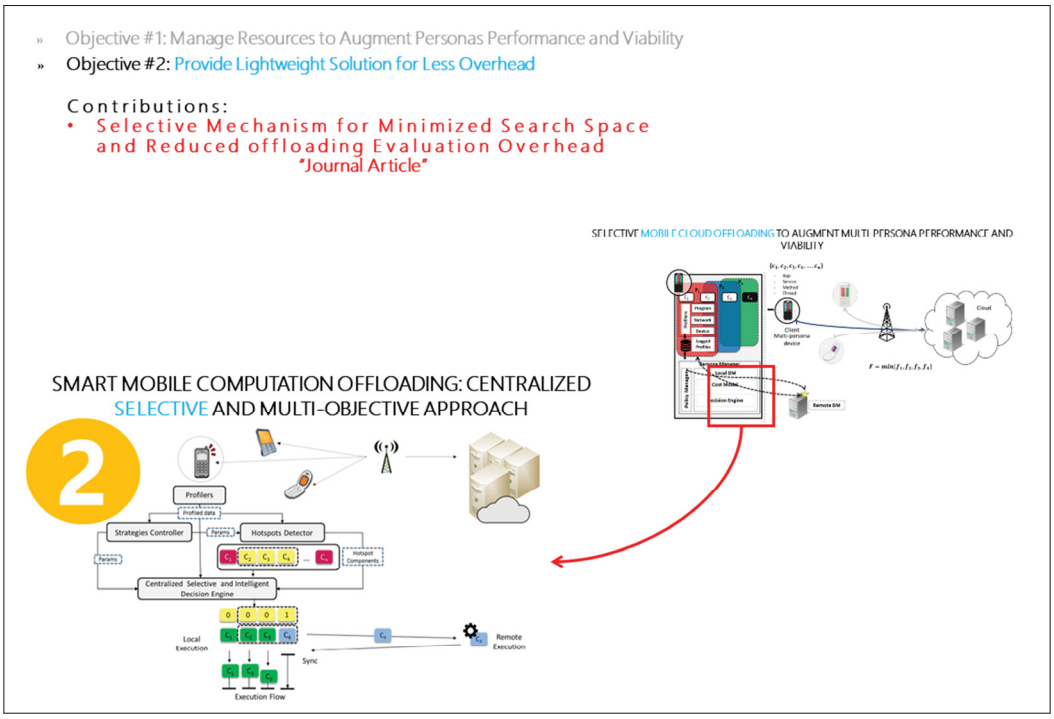


Figure 0.2 Selective approach: objectives, contributions and architecture.

The results in the first chapter reveal that the overhead of the decision maker in some scenarios is higher than the the services overhead; therefore, in the second article (Chapter 3), we address the overhead of the offloading cost model evaluation, which can create itself a bottleneck on the end mobile device with limited resources. We propose in this regard a selective approach as shown in Figure 0.2 with the following sub-objectives:

- Propose a novel selective mechanism that minimizes the search space and significantly reduces the overhead of offloading decision evaluation.

- Offer an intelligent remotely centralized decision engine able to find the best dissemination of tasks with minimal evaluation cost.

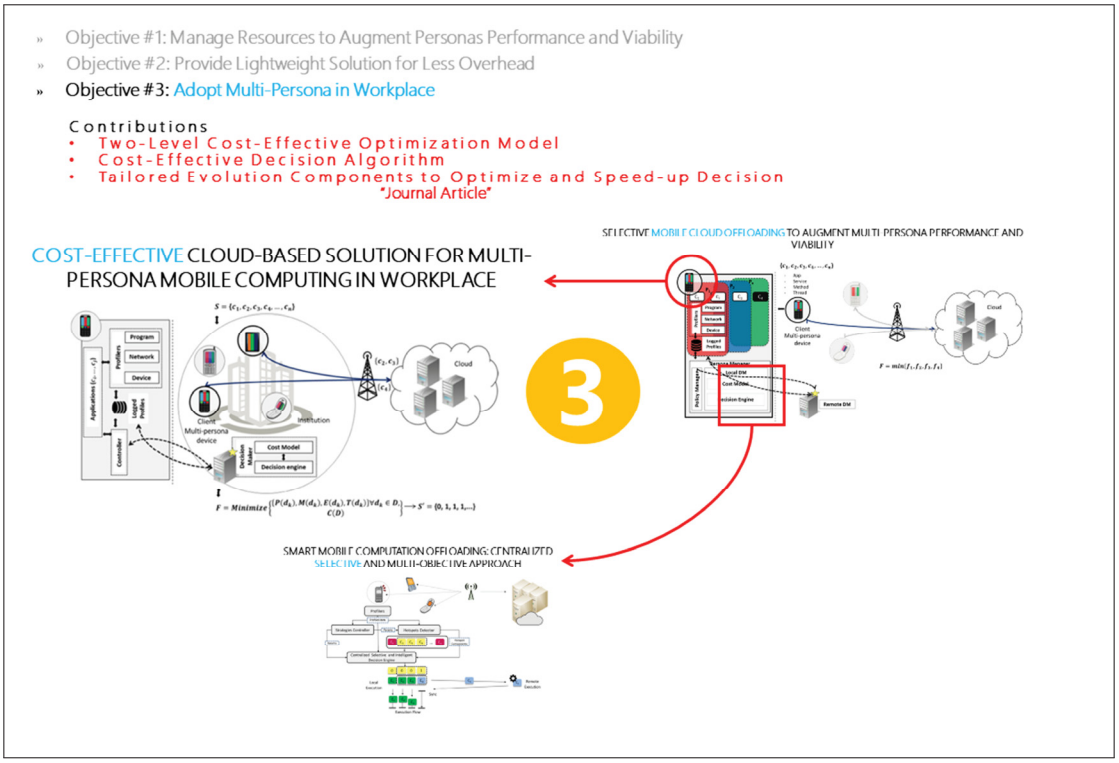


Figure 0.3 Cost-effective approach: objectives, contributions and architecture.

From different perspective, embracing such cloud-based solutions in workplace where multi-persona mobile computing is mainly introduced raises other concerns that cannot be neglected. The first concern is how to adequately disseminate the execution of services between local and remote processing in order to enforce as many personas as possible on all users' terminals engaged rather than a solo device. Second, leveraging commercial cloud resources is not free of charge and with several offloading requests generated from numerous mobile devices, the

fees aspect becomes a key factor in the equation. Therefore in the third article (Chapter 4), we aim to answer the important research question that arises when adopting such solutions in workplace. Particularly, how to balance in one hand, the usage of cloud-based offloading services to minimize processing, memory, energy and execution time in personas on as many devices engaged in an organization as possible, and on the other hand, the remote execution fees imposed on the institution itself? Accordingly, we propose a cost-effective approach as illustrated in Figure 0.3 with the following sub-objectives:

- Propose a two-level cost-effective optimization model to balance processing, memory, energy and performance of personas on different devices with minimal remote resources usage fees.
- Evaluate both centralized and decentralized decision making approaches to examine their engagement to address the raised concerns from different perspectives.

Technically, some computations might not be offloadable based on their type, security level or even their need to call device-related functionalities and hence offloading these components is not an option. Also idle apps and/oidle virtual phones consume some of the device resources and offloading would even consume more. Therefore finally, in the fourth article (Chapter 5), we examine why only offloading is not enough as a resource management solution, why proactivity is needed and what kind of advanced manageability strategies are needed beside offloading? For this end, we place the following sub-objectives in order to offer proactive and advanced solution as illustrated in Figure 0.4:

- Provide proactive solution to predict resource needs and hence avoid system crash.
- Propose advanced strategies to manage virtual phones.

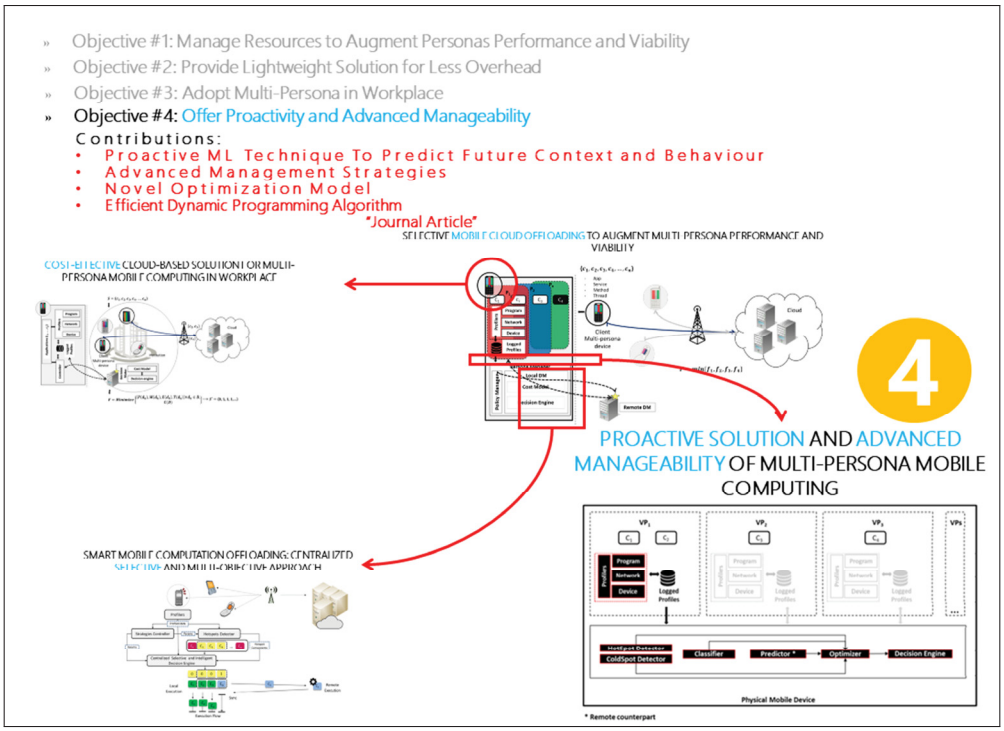


Figure 0.4 Proactive advanced approach: objectives, contributions and architecture.

- Present novel optimization model to meet the resource needs, enhance the performance on the end device and support the proposed strategies.

- Elaborate an efficient algorithm to find the adequate strategies to be applied by the end terminal.

It is worth to mention that throughout this thesis, we use "optimality" and "best distribution" in order to define a solution that can offer best "trade-off" between the optimization metrics in hand.

0.5 Technical Contributions

Through this thesis, we were able to offer the following contributions:

- A Mobile cloud-based architecture for efficient multi-persona mobile computing support.
- Cost-effective solution for multi-persona mobile computing in workplace.
- Generic, adaptable and lightweight optimization techniques for virtual phones' resource and performance management.
- Proactive method with advanced manageability strategies for efficient control of virtual phones' performance and viability.
- Competitive algorithms that automatically generate the adequate strategies to be applied by the end terminal.

0.6 Publications

- Tout, H., Talhi, C., Kara, N. & Mourad, A. (2015). Towards an offloading approach that augments multi-persona performance and viability. *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pp. 455–460.
- Tout, H., Talhi, C., Kara, N. & Mourad, A. (2016). Selective Mobile Cloud Offloading to Augment Multi-Persona Performance and Viability.*Cloud Computing, IEEE Transactions on*.
- Tout, H., Talhi, C., Kara, N. & Mourad, A. (2017). Smart mobile computation offloading: Centralized selective and multi-objective approach. *Expert Systems with Applications*, 80, 1-13.
- Tout, H., Mourad, A., Kara, N. & Talhi, C. Cost-Effective Cloud-Based Solution for Multi-Persona Mobile Computing in Workplace (Journal Article Under Review)
- Tout, H., Kara, N., Talhi, C. & Mourad, A. Proactive Solution and Advanced Manageability of Multi-Persona Mobile Computing (Journal Article Under Review).

0.7 Thesis Organization

Since this work is done on articles-basis, we start with a general literature review then we detail each of our publications in different chapter. Finally, in the last part, we conclude the thesis and draw some future directions out of remaining open research questions.

LITERATURE REVIEW

In this part, we review different approaches relevant to various aspects subject to this thesis to formulate a general context for this research work . Yet, we leave the analysis of these approaches to each chapter separately, where we compare our propositions with existing works based on different aspects relevant to each contribution.

1.1 Mobile Virtualization

With the advanced hardware of smart phones, researchers have been able to bring virtualization to mobile terminals. As a trade-off between system-level virtualization [Barr *et al.* (2010); Dall & Nieh (2013)] and user-level isolation [Inc.], researchers have proposed OS-level virtualization which shares the kernel layer to run multiple virtual instances on a single operating system [Andrus *et al.* (2011); Chen *et al.* (2015c)]. Using isolation methods that leverage namesapces at multiple levels of the mobile platform, OS-level virtualization is able to ensure that the virtual environments are completely independent, secure from each other and any failure that might occur in one of them will not affect the others. These properties are what make such architecture the most adequate for building multi-persona [Andrus *et al.* (2011); Chen *et al.* (2015c)].

Wessel et al. present a lightweight isolation mechanism for Android with access control policies, to separate one or more Android userland instances from a trustworthy environment [Wessel *et al.* (2013)]. The proposed architecture provides userspace containers to isolate and control the resources of single application or groups of applications running on top of one kernel.

Another approach is Cells [Andrus *et al.* (2011)], which enables multiple virtual phones (VPs) to run simultaneously on the same physical smartphone. It uses device namespaces to multiplex access among VPs to kernel interfaces and hardware resources such that VPs can run side-by-side in virtual OS sandboxes.

Chen et al. have proposed Condroid [Chen *et al.* (2015c)], a lightweight virtualization architecture that allows creating multiple personas by virtualizing identifiers and hardware resources on a shared OS kernel. The proposed architecture leverages namespaces for resource isolation and cgroups feature for resource control. Together, they allow Condroid to run multiple independent and securely isolated virtual instances on the same physical device.

Discussion: Throughout this thesis, it is demonstrated that even the latest lightweight virtualization techniques not only impose significant overhead on the mobile hardware, which has limited computation capabilities, memory capacity and battery lifetime, but also might force the whole system to shutdown.

1.2 Mobile Computation Offloading

Many approaches have proposed mobile computation offloading techniques to support mobile devices. These approaches have indeed proved their ability to enhance the applications performance and minimize the energy consumption on mobile devices. In mCloud framework [Zhou *et al.* (2016)], different cloud resources are considered; mobile ad-hoc device cloud, cloudlets and public cloud. The work aims to find where tasks should be executed so that the overall energy consumption and execution time is the lowest among all cloud resources in the mobile cloud infrastructure based on the current state of the device.

MAUI [Cuervo *et al.* (2010)] is an offloading framework that has been proposed by Cuervo et al. in order to reduce the energy consumption of mobile applications. The framework consists of a proxy server responsible of communicating the method state, a profiler that can monitor the device, program and network conditions, and a solver that can decide whether to run the method locally or remotely. MAUI uses its optimization framework to decide which method to send for remote execution based on the information gathered by the profiler. The results show the ability of MAUI to minimize the energy consumption of a running application.

CloneCloud [Chun *et al.* (2011)] is another offloading approach that has been presented in order to minimize the energy consumption and speed-up the execution of the running application.

A profiler collects the data about the threads running in this application and communicates the gathered data with an optimization solver. Based on cost metrics of execution time and energy, the solver decides about the best partitioning of these threads between local and remote execution. This approach does not require modification of the original application since it works at the binary level. The experiments of CloneCloud showed promising results in terms of minimizing both execution time and energy consumption of an application. However, only one thread at a time can be encapsulated in a VM and migrated for remote execution, which diminishes the concurrency of executing the components of an application.

Relying on distributed shared memory (DSM) systems and virtual machine (VM) synchronization techniques, COMET [Gordon *et al.* (2012)] enable multithreaded offloading and overcomes the limitations of MAUI and CloneCloud, which can offload one method/thread at a time. To manage memory consistency, a field-level granularity is used, reducing the frequency of required communication between the mobile device and the cloud.

Kemp has followed a different strategy and proposed Cuckoo [Kemp (2014)] that assumes computation-intensive code to be implemented as an Android service. The framework includes sensors to decide, at runtime, whether or not to offload particular service since circumstances like network type and status and invocation parameters of the service call on mobile devices get changed continuously, making offloading sometimes beneficial but not always. Cuckoo framework has been able to reduce the energy consumption and increase the speed of computation intensive applications.

Chen et al. [Chen *et al.* (2012)] have proposed a framework that automatically offloads heavy back-end services of a regular standalone Android application in order to reduce the energy loss and execution time of an application. Based on a decision model, the services are offloaded to an Android virtual machine in the cloud.

An offloading-decision making algorithm that considers user delay-tolerance threshold has been proposed by Xia et al. [Xia *et al.* (2014)]. The tool predicts the average execution

time and energy of an application when running locally on the device, then compares them to cloud-based execution cost in order to decide where the application should be executed.

ThinkAir [Kosta *et al.* (2012)] has been introduced as a technique to improve both computational performance and power efficiency of mobile devices by bridging smartphones to the cloud. The proposed architecture consists of a cloud infrastructure, an application server that communicates with applications and executes remote methods, a set of profilers to monitor the device, program, and network conditions, and an execution controller that decides about offloading. ThinkAir applies a method-level code offloading. It parallelizes method execution by invoking multiple virtual machines (VMs) to execute in the cloud in a seamless and on-demand manner achieving greater reduction in execution time and energy consumption.

Shi et al. have presented COSMOS system [Shi *et al.* (2014)] with the objective of managing cloud resources to reduce their usage monetary cost while maintaining good offloading performance. Through a master component, COSMOS collects periodically information of computation tasks and remote VMs workloads. Based on the gathered information, COSMOS is able to control the number of active VMs over time. Particularly, whenever VMs are overloaded, the system turns on new instance to handle the upcoming requests. It can also decide to shut down unnecessary instances to reduce the monetary cost in case the rest are enough to handle the mobile devices requests.

Chae et al. [Chae *et al.* (2014)] have proposed CMcloud, a new scheme that aims to maximize the throughput or minimize the server cost at cloud provider end by running as many mobile applications as possible per server and offer the user's expected acceleration in the mobile application execution. CMcloud seeks to find the least costly server which has enough remaining resources to finish the execution of the mobile application within a target deadline.

Discussion: These offloading approaches were indeed able to enhance the performance of the mobile device and reduce its energy consumption. Based on their promising results, we build an mobile cloud approach to boost the performance of the virtual instances and ensure longer viability of the physical device. However, the techniques proposed in these approaches

are not enough to reach our main goal. Therefore, our approach differs from these works in different aspects that we discuss in details separately in each chapter based on the latter contributions. To summarize, this thesis offers novel multi-objective optimization model that considers critical additional metrics relevant to the device resources. In addition, the model is generic enough to be applied at any offloading granularity and adaptable to different execution contexts. Further, it also considers the tradeoff between the device resources, the applications performance in the running personas, as well as the cost entailed by practical adoption of such cloud-based solution. In this context, this thesis is the first to deal with all these metrics which are discussed further in the next chapters.

1.3 Predictive Management Techniques for Virtual Environments

Sharing a single physical end terminal between several virtual machines raises many problems more critically, autonomic load balancing of resources. In this context, different approaches have been proposed to predict physical machine loads. Predicting future load enables proactive consolidation of VMs on the overloaded and under-loaded physical machines [Farahnakian *et al.* (2015)]. In [Farahnakian *et al.* (2013a)] and [Farahnakian *et al.* (2013b)], the authors have proposed regression methods to predict CPU utilization of a physical machine. These methods use the linear regression and the K-nearest neighbor (KNN) regression algorithms, respectively, to approximate a function based on the data collected during the lifetimes of the VMs. The formulated function is then used to predict an overloaded or an under-loaded machine. A linear regression based approach has been implemented by Fahimeh Farahnakian [Farahnakian *et al.* (2013a)]. The CPU usage of the host machine is predicted on the basis of linear regression technique and then live migration process was used to detect under-utilized and over-utilized machine. Bala et al. [Bala & Chana (2016)], have proposed a proactive load balancing approach that based on a prior knowledge of the resource utilization parameters and gathered data, machine learning techniques are applied to predict future resource needs. Various approaches have been studied such as KNN, Artificial Neural Network (ANN), Support Vector Machines (SVM) and Random Forest (RF). The approach having maximum accuracy

has been utilized as prediction-based approach. Xiao et al. [Xiao *et al.* (2013)] have also used a load prediction algorithm to capture the rising trend of resource usage patterns and help identifying hot spots and cold spots machines. After predicting the resource needs, Hot spot and cold spot machines are identified. When the resource utilization of any physical machine is above the hot threshold, the latter is marked as hotspot. If so, some VMs running on it will be migrated away to reduce its load [Xiao *et al.* (2013)]. On the other hand, cold spot machines either idle or having the average utilization below particular threshold, are also identified. If so, some of those physical machines could potentially be turned off to save energy [Xiao *et al.* (2013)] [Beloglazov & Buyya (2010)].

Discussion: While offloading can augment the device resources and boost the applications performance, it might fail to avoid system crash in some scenarios. When the needed resources are higher than those available on the device and involved applications cannot be offloaded whether because they are native tasks or according to their criticality level, additional proactive technique and more advanced manageability strategies become a must. Approaches presented in this section show that estimating resource needs enables proactive consolidation of VMs on a single physical machine in a cloud environment, the same concept applied on multi-persona mobile devices. In this thesis, future context and resource requirements are predicted beforehand through machine learning techniques in order to take proactive actions that aim to avoid performance degradation and system crash on the mobile device. Further, beside offloading, additional management strategies are advanced including personas switch-off and applications shut down, all applied based on the device state, classification scheme and predicted future context.

1.4 Dynamic Offloading Algorithms

Different algorithms have been proposed in order to find for each component, whether it should be offloaded or executed locally on the device. A linear program solver has been adopted by [Cuervo *et al.* (2010); Chun *et al.* (2011)], while dynamic programming algorithms have been studied in other works. A Dynamic Programming (DP) algorithm was proposed in [Liu & Lee

(2014)] in order to determine what to offload. However, a backtracking algorithm was needed to find the final decisions, which was time consuming. A dynamic offloading algorithm based on Lyapunov optimization was presented in [Huang *et al.* (2012)]. The algorithm is based upon a relationship between the current solution and the optimal solution requiring a considerable amount of execution time and many iterations to converge upon a solution [Shahzad & Szymanski (2016)]. Another dynamic programming approach has been proposed in [Toma & Chen (2013)], yet it doesn't consider the energy consumed in the mobile device which is an important criteria for resource constrained mobile devices. A semidefinite relaxation approach for the offloading problem was presented by Chen et al., [Chen *et al.* (2015a)]. Their work considered a mobile cloud computing scenario consisting one nearby computing access point, and a remote cloud server(s). Their proposition is based on an algorithm that first solves a linear program, and uses randomization and relaxation to generate an integer solution. The algorithm can find a near-optimal solution when using about 100 trials of relaxation. In [Shahzad & Szymanski (2016)], a dynamic programming algorithm with a hamming distance (DPH) is proposed. The algorithm generates periodically random bit strings of 0s and 1s, for remote and local execution of tasks, and utilize sub-strings when they improve the solution. The algorithm can find a nearly optimal solution after several iterations. The authors use a Hamming distance criterion to terminate the search process and hence obtain the final decision quickly. The stopping criterion is met when a given fraction of tasks are offloaded.

Discussion: Different algorithms have been leveraged in the literature to evaluate and take an offloading decision. In this work, we prove that the problem we are addressing is NP-hard and hence we first propose an algorithm based on heuristics. As in the above approaches, also dynamic programming algorithms were able to prove their ability to generate an efficient offloading strategy in a reasonable time, we study in our turn its efficiency in multi-persona context by proposing a dynamic programming-based algorithm and comparing it to heuristic-based approach. Under different constraints, these algorithms try to find the adequate strategies to be applied by each persona and its components in a way to offer a trade-off between all the objective functions formulated in the optimization cost model.

1.5 Conclusion

We reviewed in previous sections, existing approaches relevant to mobile virtualization, computation offloading, proactive management techniques for virtual environments and dynamic offloading algorithms. As discussed above, though the efficiency of the existing propositions, many technical limitations need to be addressed in order to meet with multi-persona requirements on resource-constrained mobile devices. Each of the following chapters is devoted to tackle set of multi-persona issues and provide part of the full solution proposed in this thesis.

CHAPTER 2

ARTICLE 1: SELECTIVE MOBILE CLOUD OFFLOADING TO AUGMENT MULTI-PERSONA PERFORMANCE AND VIABILITY

Hanine Tout[1], Chamseddine Talhi[1], Nadjia Kara[1], Azzam Mourad[2]

[1] Departement of Software Engineering and IT, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Department of Computer Science and Mathematics, Lebanese American University, 1102 2801 Chouran Beirut, Lebanon

Published in IEEE Transactions on Cloud Computing
(DOI: 10.1109/TCC.2016.2535223)

2.1 Abstract

Fueled by changes in professional application models, personal interests and desires and technological advances in mobile devices, multi-persona has emerged recently to keep balance between different aspects, in our daily life, on a single mobile terminal. In this context, mobile virtualization technology has turned the corner and currently heading towards widespread adoption to realize multi-persona. Although recent lightweight virtualization techniques were able to maintain balance between security and scalability of personas, the limited CPU power and insufficient memory and battery capacities, still threaten personas performance and viability. Throughout the last few years, cloud computing has cultivated and refined the concept of outsourcing computing resources, and nowadays, in the coming age of smartphones and tablets, the prerequisites are met for importing cloud computing to support resource constrained mobiles. From these premises, we propose in this paper a novel offloading-based approach that based on global resource usage monitoring, generic and adaptable problem formulation and heuristic decision making, is capable of augmenting personas performance and viability on mobile terminals. The experiments show its capability of reducing the resource usage overhead and energy consumption of the applications running in each persona, accelerating their execution and improving their scalability, allowing better adoption of multi-persona solution.

Index Terms: Multi-Persona, Mobile device, Mobile virtualization, Mobile Cloud Computing, Offloading, Multi-Objective Optimization, Heuristic algorithms.

2.2 Introduction

In today's high-tension and fast-paced world, technological advances have changed the concept of mobile devices from primitive gadget to full computers that accommodate work, personal and mobility needs. Out of this wave, BYOD (Bring Your Own Device) revolution has emerged across a variety of industries, as a policy to allow end-users to use personally owned mobile devices for business tasks [Rouse (2012a)]. However, clearly having personally owned devices accessing corporate data and apps raise a number of risks and security concerns. One solution is COPE (Corporate-Owned, Personally-Enabled), a business model in which employees use corporate issued devices [Rouse (2014)]. Yet, by granting manageability to enterprises, employees are sacrificing both usability and privacy. Another solution is to carry two mobile devices, but the natural tendency for most people is to combine professional and personal needs on the same physical device, hence again not a good solution. As an alternative winning technology, mobile devices with dual persona were released enabling two phones-in-a-phone, one for private personal use and another for business use [Rouse (2012b)].

However, nowadays, multi-persona has become the name of the game. Customizing isolated personas for banking services, e-commerce, corporate data, social networking and games, allow parents to efficiently manage financial transactions, prevent untrusted applications from accessing critical information and share the device with children without ending up with accidental phone calls, unintended in-app purchases or even access to restricted content [Andrus *et al.* (2011)]. Also, traveling for a conference or attending a trade show is a very common practice for businesspersons, who are not tied anymore with just corporate and personal personas. An additional persona customized by the event coordinator to push relevant apps, files, feeds, agenda, maps and tourism guides, offers better management and seamless access to event resources [Spaces]. Further, multi-persona proves its utility in other areas where neither carrying multiple mobile devices nor complying with single policy is a choice. While working

at their private clinic and at multiple hospitals, doctors are subject to different mobile policies, reflecting each of the different institutions. Carrying multiple devices to accommodate with different systems drains their productivity. Whereas with personal, clinic and hospitals personas, multi-persona allow doctors to comply with the policy of each and effectively treat their patients while maintaining their own unburdened personal use of the device [Eiferman (2014b)]. Whether in these or any other example, the success of multi-persona lies in its capability of consolidating multiple mobile devices on a single terminal, while making the latter able to clearly distinguish between the different contexts in which it is used.

Mobile virtualization is one of the key technologies applied to realize multi-persona. Similar to virtualization on servers and desktop machines, mobile virtualization allows to create multiple virtual environments that live alongside on a single terminal, where in this case, the latter is a mobile device and the environments are called personas. Yet, to realize multi-persona, mobile virtualization is much more challenging since it requires a trade-off between secure isolation and scalability of personas on mobile devices with limited resources. Therefore, in the last few years, researchers have proposed lightweight virtualization techniques [Wessel *et al.* (2013); Andrus *et al.* (2011); Chen *et al.* (2015c)] towards mitigating the virtualization overhead on mobile terminals while keeping a certain level of isolation between the virtual environments. Nevertheless, even with these techniques, the limited CPU power and insufficient memory and battery, threaten personas performance and viability at any time being. Our experiments in Section 2.4 show drastic increase in the CPU usage, energy consumption and execution time of the running applications. Even worse, because of lack of memory, the personas are forced to shut down under certain circumstances. These severe problems call for the integration of new techniques capable of augmenting personas performance and viability.

A lot of attention has been given recently to mobile cloud computing, which imports new cloud computing services, applications and infrastructures to support mobile devices [Khan *et al.* (2014); Abolfazli *et al.* (2014); Zhou *et al.* (2015)]. In order to address the resource limitations of mobile platforms, many researchers have proposed offloading techniques [Hung *et al.* (2012); Cuervo *et al.* (2010); Kosta *et al.* (2012); Kemp (2014); Chen *et al.* (2012); Chun *et al.*

(2011)] to migrate computation intensive components to be executed on resourceful infrastructure. While these approaches are proposed to optimize single application, running multiple apps in multi-persona implies resource profiling and offloading evaluation to be done solely for each app, which impose high overhead on the mobile terminal. Different offloading approach [Mazza *et al.* (2014)] has been proposed towards optimizing the execution of applications on multiple mobile devices. Yet the proposed technique is not able to identify components to be offloaded but rather generates only their fraction/percentage.

We propose in this article an offloading-based approach to augment multi-persona performance and viability on resource constrained mobile devices. Our proposition consists first of monitoring the components running in each persona using per persona profiler, and then determining their optimal execution environment based on a generic and adaptable decision model. Taking into account four conflicting objectives of minimizing CPU and memory usages, energy consumption and execution time, we formulate the decision model as multi-objective optimization problem, generic enough to be applied on any components unit (i.e., applications, services, methods and threads) and adaptable to different execution settings. Using heuristics to solve this latter, our approach dictates for each component whether it should be executed locally or offloaded for remote execution.

The main contributions of our approach are threefold:

- Proposing offloading-based architecture to augment multi-persona performance and viability on mobile devices.
- Providing generic and adaptable multi-objective optimization model to formulate multi-persona problems independently of the offloading granularity and adapt offloading evaluation to different execution contexts.
- Generating exact optimal distribution of multi-persona components through heuristics.

The roadmap of the paper is as follows. In Section 2.3, we present relevant background information and we study existing related works. In Section 2.4, we highlight the problems caused

by running multiple personas on a single mobile device, while in Section 2.5, we illustrate our proposed solution to address these issues. In Section 2.6, we present our multi-objective optimization model for offloading evaluation and its complexity analysis whereas in Section 2.7, we describe the heuristic algorithm to solve it. Later, in Section 2.8, we provide details about the implementation as well as the experimental results that prove the efficiency of our proposition. Finally, in Section 2.9, we conclude the paper and draw our future research directions.

2.3 Background and Related Work

We present in this section some background information about mobile virtualization and offloading and the relevant state of the art review.

2.3.1 Mobile Virtualization

As smartphones and tablets are growing more and more sophisticated, researchers were able to bring virtualization to such mobile terminals. System-level virtualization [Barr *et al.* (2010); Dall & Nieh (2013)] is a technique that offers the ability to run multiple virtual environments on one physical device using an additional software layer called a hypervisor (or microkernel) [Wessel *et al.* (2013)]. The virtual environments may run the same or even different operating systems. Even though this technique offers full isolation between the virtual environments, it suffers from significant overhead due to the complete software stack in each instance (i.e., Kernel, Middleware and Apps) [Andrus *et al.* (2011)]. Therefore, when it comes to more than just two personas on the device, this architecture will not be the right choice to go. Per contra, the user-level isolation technique [Inc.] reaches separation by wrapping applications instead of creating virtual environments, which makes it very lightweight when applied on mobile devices. However, by keeping separation just at the applications level, critical, malicious, personal, business and any other type of applications will be running in the same environment. This makes user-level isolation a bandage more than a real solution that can realize multi-persona which requires much higher security [Eiferman (2013)].

As a trade-off between both above techniques, researchers have proposed recently OS-level virtualization, also called container-based virtualization, which is a technique that shares the kernel layer to run multiple virtual instances on a single operating system [Andrus *et al.* (2011); Chen *et al.* (2015c)]. Allocating a minimum set of resources for each instance, make the available OS resources enough for running more than just two personas on top of it. Also using isolation techniques that leverage namesapces at multiple levels of the mobile platform, this technique is able to ensure that the virtual environments are completely independent, secure from each other and any failure that might occur in one of them will not affect the others. These properties are what make such architecture the most adequate for building multi-persona [Andrus *et al.* (2011); Chen *et al.* (2015c)]. Wessel et al. present a lightweight isolation mechanism for Android with access control policies, to separate one or more Android userland instances from a trustworthy environment [Wessel *et al.* (2013)]. The proposed architecture provides userspace containers to isolate and control the resources of single application or groups of applications running on top of one kernel. Another approach is Cells [Andrus *et al.* (2011)], which enables multiple virtual phones (VPs) to run simultaneously on the same physical smartphone. It uses device namespaces to multiplex access among VPs to kernel interfaces and hardware resources such that VPs can run side-by-side in virtual OS sandboxes. Lately, Chen et al. have proposed Condroid [Chen *et al.* (2015c)], a lightweight virtualization architecture that allows creating multiple personas by virtualizing identifiers and hardware resources on a shared OS kernel. The proposed architecture leverages namespaces for resource isolation and cgroups feature for resource control. Together, they allow Condroid to run multiple independent and securely isolated virtual instances on the same physical device.

Despite the lightweight approaches, running multiple personas on the same physical mobile terminal remains challengeable. The limited CPU, memory capacity and battery power, all threaten the performance of the running personas and make the device unable to tolerate their survivability. While varying the number of personas, number and type of applications running in each, our experiments in Section 2.4 show drastic increase in the CPU and memory usages, energy consumption on the device as well as in the execution time of the running applications.

Table 2.1 Taxonomy of mobile code offloading approaches.

Criteria / Technique	Target	Granularity	Decision Model	Cloud Features	Gain
[Hung *et al.* (2012)]	One App on Single Device	Application	Does Not Apply	Virtual Phone	Faster Execution
[Cuervo *et al.* (2010)]		Method	Energy	Server	Faster Execution and Energy Saving
[Kosta *et al.* (2012)]		Method	Energy and Time	Dynamic Allocation and Management of VMs	
[Kemp (2014)]		Service	Energy and Time	Server	
[Chen *et al.* (2012)]		Service	Time, Energy and Battery Level	Virtual Phone	
[Chun *et al.* (2011)]		Thread	Energy and Time	Server	
[Mazza *et al.* (2014)]	Multiple Apps from Multiple Devices	Service	Energy and Time	Server	Distribution Fraction, Faster Execution and Energy Saving
Our Approach	Multiple Apps from Multi-Persona	Generic	Energy, Time, CPU and Memory	Server	Exact Distribution, Faster Execution, Energy Saving, Minimized CPU and Memory Usages

They reveal also the inability of the device to run more than three personas. Even though we are able to start a fourth persona, they all shut down once a new application starts executing in this latter.

2.3.2 Offloading

In turn, mobile cloud computing has brought cloud computing capabilities to support mobile devices, ranging from outsourcing software and platforms all the way to infrastructure [Ahmed *et al.* (2015b,c,a); Abolfazli *et al.* (2014)]. Different offloading concepts exist in this context. The explosion of mobile internet applications, like multimedia newspapers, social networking services, audio and video streaming, is the main reason behind the significant overload on the cellular networks. In this context, traffic offloading [Fiandrino *et al.* (2015); Han *et al.* (2010); Andreev *et al.* (2014)] has been proposed, which is the use of complementary networks like Wi-Fi for data transmission in order to reduce the data carried on the cellular network. On the other hand, the limited resources of mobile devices have triggered another research domain of computation offloading, which has different concept and objective compared to traffic offloading. In computation offloading, resource-hungry applications, services, methods or threads are offloaded out of the device to be executed on resource-rich and more powerful infrastructure like remote servers. These components are profiled and an offloading decision is taken based

on predefined optimization metrics that determine the cost-benefit of offloading. Our proposition is based on computation offloading, therefore in Table 2.1, we provide a classification of existing relevant techniques to better position our work. Target indicates what the offloading techniques aim to optimize, granularity identifies the type of components where offloading is applied and the decision model defines the metrics for offloading evaluation. Gain shows the benefits of each technique on the mobile terminal and cloud features are the assets used to attain these gains.

From Table 2.1, approaches aiming to optimize an application execution on single mobile device can be further distinguished based on their granularity:

Application-based offloading: Hung et al. [Hung *et al.* (2012)] have proposed an approach to execute mobile applications in a cloud-based virtualized environment. The proposed architecture consists of a mobile device connected to virtual phone in the cloud, and an agent program installed on the device whose purpose is to allocate a delegate system on the cloud and communicate the application status. This work presents an application-level migration using the pause/resume concept in android. The application is copied in case it does not exist on the virtual phone. Otherwise, the agent triggers OnPause function of the application and sends its state to the remote agent where it get resumed using OnResume function. Their approach has been able to prove its ability to offload applications out of the mobile device, to be executed on virtual phone in the cloud. However, what is missing in their proposition is the criteria, objective, or circumstances under which a certain application should be considered for offloading.

Method-based offloading: MAUI [Cuervo *et al.* (2010)] is an offloading framework that aims to reduce the energy consumption of mobile applications. The framework consists of a proxy server responsible of communicating the method state, a profiler that can monitor the device, program and network conditions, and a solver that can decide whether to run the method locally or remotely. MAUI uses its optimization framework to decide which method to send for remote execution based on the information gathered by the profiler. The results show the ability of MAUI to minimize the energy consumption of a running app. ThinkAir [Kosta *et al.* (2012)]

aims to improve both computational performance and power efficiency of mobile devices by bridging smartphones to the cloud. The proposed architecture consists of a cloud infrastructure, an application server that communicates with applications and executes remote methods, a set of profilers to monitor the device, program, and network conditions, and an execution controller that decides about offloading. ThinkAir applies a method-level code offloading. It parallelizes method execution by invoking multiple virtual machines (VMs) to execute in the cloud in a seamless and on-demand manner to achieve greater reduction in execution time and energy consumption. ThinkAir was also able to demonstrate its capability in that regard.

Service-based offloading: Cuckoo [Kemp (2014)] is another offloading framework that follows a different strategy for offloading computation-intensive tasks. As precondition, all compute intensive code should be implemented as an Android service. The framework includes sensors to decide, at runtime, whether or not to offload particular service since circumstances like network type and status and invocation parameters of the service call on mobile devices get changed continuously, making offloading sometimes beneficial but not always. Cuckoo framework has been able to reduce the energy consumption and increase the speed of computation intensive applications. Chen et al. [Chen *et al.* (2012)] have proposed another framework that follows similar strategy to automatically offload heavy back-end services of a regular standalone Android application. Yet, based on a decision model, the services are offloaded to an Android virtual machine in the cloud. Their proposition has not been implemented and evaluated yet.

Thread-based offloading: CloneCloud [Chun *et al.* (2011)] is a system that aims to minimize both execution time and energy consumption of a running application. It consists of a profiler, which collects the data about the threads running in this app and communicates the gathered data with an optimization solver. Based on cost metrics of execution time and energy, the solver decides about the best partitioning of these threads between local and remote execution. This approach does not require modification in the original application since it works at the binary level. The experiments of CloneCloud showed promising results in terms of minimizing both execution time and energy consumption of an application. However, only one thread at a

time can be encapsulated in a VM and migrated for remote execution, which diminishes the concurrency of executing the components of an application.

On the other hand, Mazza et al. [Mazza *et al.* (2014)] proposes an approach that aims to optimize the execution of applications in a system that involves multiple mobile devices. In their work, a partial offloading technique has been proposed for heterogeneous networks infrastructure (HetNets). Depending on the number of devices connected in this network and constrained by both the energy consumption and execution time, the proposed approach is able to generate the percentage of tasks to be offloaded, aiming to optimize the entire system rather than just a single device.

2.3.3 Proposed Approach Positioning

Computation offloading requires device status, network conditions and applications to be monitored, in order to study the effectiveness of offloading when a decision should be made. The gathered information formulate the input of a solver that evaluates the decision model metrics to decide whether a component is to be offloaded or executed locally. Existing approaches [Hung *et al.* (2012); Cuervo *et al.* (2010); Kosta *et al.* (2012); Kemp (2014); Chen *et al.* (2012); Chun *et al.* (2011)] are proposed for single application where profiling and offloading evaluation are done solely for each app. Therefore, with multiple applications on the mobile terminal, a profiler and a solver should be dedicated for each, which cause significant overhead in multi-persona where many independent components from different applications are running on the device. Multi-persona necessitates higher view of resource consumption and global formulation of the decision making metrics, therefore our work provides per persona profiler, and eventually global formulation of the optimization model. The proposed model is generic enough to be applied with any offloading component unit (i.e., at any granularity level) and adaptable to different execution settings like lack of memory and low battery. The proposed approach is capable of minimizing CPU and memory usages that affect the performance as well besides the energy and execution time of the applications.

On the other hand, different work [Mazza *et al.* (2014)] exists involving multiple mobile devices, where the aim is to optimize the entire system rather than a single terminal. The proposed approach generates the percentage/fraction of components to be offloaded from the system, yet identifying what components to be offloaded is needed in order to execute the offloading process. Therefore, more effectively, our approach can generate the exact distribution of all the components running in each persona at any time being with the intent of augmenting personas performance and viability. Seeing that such valuable decision can be costly, we discuss later in the paper some alternatives that can decrease its overhead.

2.4 Problem Illustration

Table 2.2 Applications in each persona.

Weight	**Application**
Lightweight	Zip: This application creates archive folder from original files. To make it lightweight, we use files having total size of 3 MB. Unzip: The unzip application extracts the content of the archived folder created using the Zip app.
Moderate	Virus Scanning: This application scans the contents of some files on the phone against a library of 1000 virus signatures, one file at a time. To implement moderate application, we fix the size of the files to 100 KB.
Computation Intensive	NQueens Puzzle: This puzzle implements the algorithm to find all possible solutions of the typical NQueens problem, and return the number of solutions found. We consider N=13 to create computationally intensive problem.

No matter how sophisticated mobile devices are growing, they still have limited hardware in terms of computing power, memory capacity and battery lifetime. Running multiple personas on a single mobile device is yet impeded by these limitations, rendering personas performance and viability on the line [Tout *et al.* (2015)]. To shed the light on these issues that we address in this paper, we vary the number of personas running diversity of lightweight, moderate and heavy applications, and we compare the resource consumption that affect personas lifetime as well as the execution time that influence their performance. To model different usage scenarios

of the device, we consider in each persona four applications of different weights as described in Table 2.2.

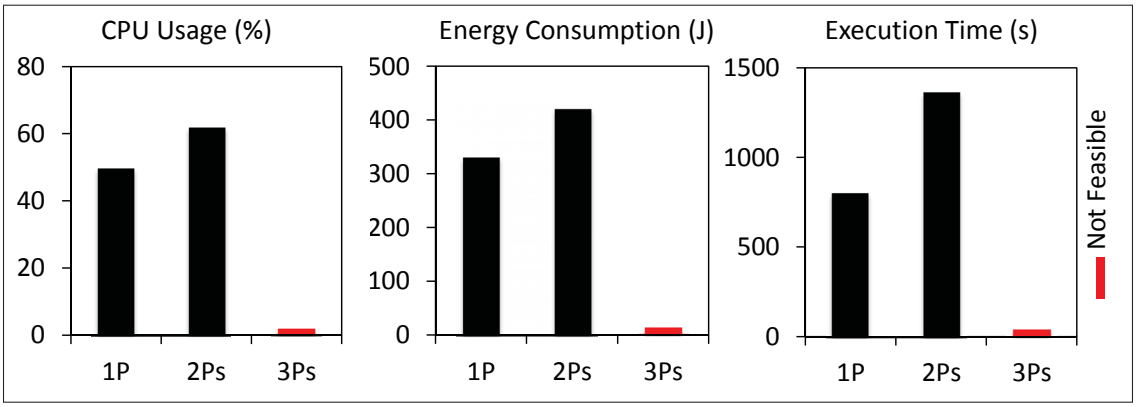


Figure 2.1 Multi-Persona efficiency and viability.

The results depicted in Figure 2.1 show drastic increase in the CPU usage that was originally 49 % with 1 Persona (1P) running the four apps, but reached 62 % with 2 Personas (2Ps). Also the energy consumption, which is consumed by the applications usage on CPU as well as the one spent on the screen, has significantly increased from 330 J to 420 J. As for the execution time, which denotes the time taken till the end of execution of the last app in each persona, it took 780 s in one persona, yet up to 1380 s with two personas. This long execution time is due to the NQueens puzzle, which we use to overload the device. Another interesting observation is in the third scenario (3Ps) where it was impossible to run the same apps in three personas, as the personas kept shutting down due to lack of memory on the mobile terminal as well as the high consumption of other resources. These results reveal the inability of the mobile device resources to afford high performing personas neither to tolerate their viability. In the light of these serious problems, it is indispensable to integrate new techniques capable of minimizing the resource consumption and execution time of different type of applications running in each persona. For details about the implementation and tools used, please refer to Section 2.8.1 which is devoted for that end.

2.5 Offloading Meets Multi-Persona

In Section 2.3, we distinguished our proposition from existing offloading approaches. In what follows we go deeper to explain it in details and highlight its contributions. The architecture of our approach is depicted in Figure 2.2 that essentially focuses on the mobile device structure, since the main dilemma lies there.

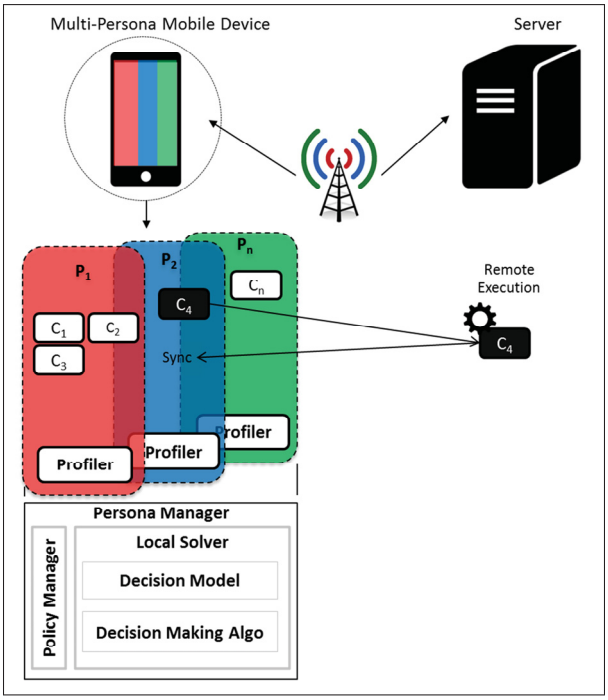


Figure 2.2 Proposed architecture.

We build our approach on top of OS-level virtualization, which is as we discussed in Section 2.3, the architecture that best fit for multi-persona solution. In each persona ($P_1 ... P_n$) we add a profiler to monitor the latter resources. Differently from the literature, this profiler is not dedicated to one application at a time neither to components belonging to one application, but rather it is devoted for the entire persona. Each profiler gathers information about CPU and memory usages, energy consumption, execution time and other relevant data for all the components running in the persona. These components ($C_1 ... C_n$) can be applications, services,

methods or even threads. The profiler examines also the connectivity availability, bandwidth and latency in the persona where it runs.

Next comes the role of the solver which uses the gathered information to construct a global decision model involving all the components running in each persona, rather than just one application on the device like in existing approaches. The decision model is based on four metrics that affect personas performance and viability, which are minimizing CPU usage, memory and energy consumptions and execution time. With these four conflicting metrics, we formulate the decision model as multi-objective optimization problem. It is worth to mention that the latter is generic enough to be independent of the offloading component unit (i.e., application, services, methods, threads). Further, this model can be automatically adapted to different execution settings. Particularly, with low CPU, memory, battery level or the need for high performing applications, the model can be adapted giving more priority for the relevant metric(s), which is/are in critical situation. We believe that this will be a valuable track in future work. Other type of adaptations may also apply, like restricting the execution of particular component(s) to the mobile device (or to remote server) for security reasons. Furthermore, the model can exclude persona(s) and/or component(s) to try whether shutting them down can be more efficient for the personas performance, and hence notify the user accordingly (e.g., component is running but has not been used for a while). All these adaptation settings can be enforced by the policy manager.

Finally, after formulating the problem, a decision algorithm and part of the solver module, is responsible of generating the exact distribution of the components running in each persona on the device. The decision dictates for each component whether it should be executed on the device or offloaded for remote execution. This is also another added value to the existing approaches, which generate only the fraction or percentage of local and remote tasks. For the solver algorithm, we use heuristics and more specifically genetic algorithms (GAs), which are able to find the optimal distribution that complies with multi-persona problem's objectives. To decrease the overhead of the decision making process, which is needed for granular offloading, one solution provided in our approach is to generate good solution rather than an optimal one.

This can offer a trade-off between components overhead and decision overhead. Whenever connectivity is not available, our approach do not call the solver but takes directly a decision to run the components locally in order to reduce the overhead caused by the solver. Also whenever certain scenario is repeated, yet with different resources availability or constraints, a delta value is to be computed in order to reduce the solver overhead. We also offer the ability to even offload the decision making process by implementing a counterpart of the solver on the server side, since based on the decision model complexity, the decision making process can be time consuming and might require additional resources.

2.6 Multi-Objective Optimization for Multi-Persona

In this section, we present a formal definition of the multi-persona problem, explain the computational analysis to prove its complexity and finally show its formulation as multi-objective optimization problem.

2.6.1 Problem Definition

Assumptions:

- Some components might not be offloadable
- Components are independent
- Network might not be stable in terms of availability, bandwidth and latency

We consider a mobile device of multiple personas $P = \{P_1, ...P_m\}$ where each of them is running a set of components $C = \{C_1, ...C_n\}$, which can be applications, services, methods or even threads that implement the applications functionalities. Each component has demands in terms of energy consumption, execution time, memory and CPU usages. Due to limited resources on the mobile device in terms of CPU, memory and battery, part/all of the components in the running personas should be offloaded for remote execution. Finding the best distribution of

these components is a complex and challenging problem. The multi-persona problem can be formulated as follows:

Problem Definition 1. *Given a set of components running in each persona P_i, where each of these components C_j has energy consumption $E^l_{c_j,p_i}$, execution time $t^l_{c_j,p_i}$, memory usage $M^l_{c_j,p_i}$ and cpu usage $CPU^l_{c_j,p_i}$ for local execution and $E^r_{c_j,p_i}$, $t^r_{c_j,p_i}$, $M^r_{c_j,p_i}$, $CPU^r_{c_j,p_i}$ for remote execution, α_{c_j,p_i} an indicator whether they are offloadable or not, network bandwidth B and latency L, find the best distribution of components between local and remote execution in a way to minimize their energy consumption, execution time, memory and CPU usages.* Minimizing the energy consumption, execution time and memory and CPU usages form the fundamental objectives that can augment performance and ensure viability of the personas running on the mobile device. Yet, this is a complex and challenging problem for the following reasons.

- First, minimizing energy consumption, execution time, memory and CPU usages are conflicting objectives, therefore finding the best tradeoff among them is not a simple task.
- Second, computing local and remote partitions of these components suffers from an exponential search space in the number of different possibilities in which these components can be distributed, which renders the problem heavy. This is similar to the various ways n distinct objects (components) can be distributed into m different bins with k_1 objects in the first bin, k_2 in the second, etc. and k_1+k_2+....k_m= n. This indeed is obtained by applying the multinomial theorem where $(k_1+k_2+...k_m)^n = \sum \binom{n}{k_1 k_2 .... k_m} k_1^{n_1} k_2^{n_2} ... k_m^{n_m}$. In our case $m = 2$ bins, one is the mobile device and the second is the remote server thus, for n components, there are 2^n different distribution possibilities.

To further emphasize the complexity of the problem, we consider the case of three personas with only four components in each. To compute the portion of the components in each persona that should be offloadded and the other that will run locally on the mobile device, there are $2^{12}(4096)$ different mapping possibilities. When this number might not appear to be that big in case the components are applications or services, it will dramatically increase when the components are methods or threads. In such case, the number of components will reach hundreds

or even thousands and it would be there 2^{100} or 2^{1000} possible distributions! which makes it hard to find their exact distribution.

Theorem 1. *Multi-Persona Multi-Objective Optimization problem is NP-Hard*

Proof: *We reduce the multi-objective-m-dimensional Knapsack Problem [Lust & Teghem (2012)] to our multi-persona problem (MPP). The idea is that if a case of the multi-persona multi-objective optimization problem can be solved, then it can be used to solve the multi-objective-m-dimensional Knapsack Problem (MOMKP). Given the MOMKP - a collection of n items* $a_1,...,a_n$, *where each item* a_i *has m weights* $w_{ki} \in \mathbb{N}, k = 1,...,m$ *and t values* $p_{ki} \in \mathbb{N}, k = 1,...,t$ *and a knapsack of m capacities* $c_k \in \mathbb{N}$, $k = 1,...,m$ *- we construct the Multi-Persona problem as follows: Setup m personas* $P = \{p_1,...,p_m\}$ *with n components in each, forming a set of* $n * m$ *denoted as x components* $C = \{C_1,...,C_x\}$, *one corresponding to each item in MOMKP.*

- *For components* C_i, *set the resource demands in terms of memory and CPU of each component as the weights of the items in the sack.* $M^l_{c_i}$, $M^r_{c_i}$, $CPU^l_{c_i}$, *and* $CPU^r_{c_i}$, *are the memory and CPU usages when* C_i *is running locally and when executed remotely, respectively.*

- *For components* C_i *set* $a_{c_i}.f_1$, $a_{c_i}.f_2$, $a_{c_i}.f_3$ *and* $a_{c_i}.f_4$ *as the values of each item, where they form the cost of each component in terms of energy consumption, execution time memory usage and CPU usage respectively. So that* $\sum_{i=1}^{x} a_{c_i}.f_j$ *where* $j = 1,...,4$ *constitute each of our objective functions correspondingly.*

With this reduction, the content of the knapsack is a portion of components, which is selected to run on the mobile device such that the total of each value (cost) is minimized (rather than maximized as in knapsack, but they are essentially the same), and vice versa, and a solution to our problem yields a solution to the MOMKP. Thus an algorithm for solving the multi-persona problem can be used to solve the multi-objective m-dimensional Knapsack problem. Hence it follows that our problem is NP-Hard.

2.6.2 Problem Formulation

Table 2.3 describes the notations used in the problem formulation.

Table 2.3 Formulas notations.

Variable	Description
m	Number of personas
p	Persona
n	Number of components in a persona
c	Component
P_{cpu}	Power consumed by the cpu when the component is executed locally
P_{sc}	Power consumed by the screen when the component is executed locally
$P_{cpu,idle}$	Power consumed by the cpu when it is idle waiting for remote results
P_{na}	Power consumed by the network when it is active
P_{tr}	Power consumed by the device during transmission
$t^l_{c,p}$	Execution time of the component c running locally
$t^r_{c,p}$	Execution time of the component c running remotely
$D_{c,p}$	Size of data exchanged between the device and the cloud for offloading c
$M^l_{c,p}$	Memory consumed by the device when c is executed locally
$M^r_{c,p}$	Memory consumed by the device when c is executed remotely
$C^l_{c,p}$	CPU usage on the device when c is executed locally
$C^r_{c,p}$	CPU usage on the device when c is executed remotely
L	Latency
B	Bandwidth
$x_{c,p}$	Decision variable that indicates whether c should be offloaded or not
$\alpha_{c,p}$	Indicates whether c is offloadable or not
a_p	Indicates whether persona p should be included in the decision process or not
$b_{c,p}$	Indicates whether c should be included in the decision process or not

1. ***Minimize Energy Consumption*** The total energy consumption is equal to the one consumed on local components plus the one for offloaded components. When running components locally, they consume energy on the CPU processing and screen brightness. As to execute components remotely, the energy is spent on the CPU being idle, screen brightness and network active while waiting for the remote execution. In addition it consists also of

the energy consumed by the device for data transmission (i.e., upload and download).

$$F_1 = min\left[\sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}(1-x_{c,p})\Big((P_{cpu}+P_{sc})\times t^l_{c,p}\Big) + \sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}x_{c,p}\Bigg(\Bigg(\Big((P_{cpu,idle}+P_{sc}+P_{n,a})\times t^r_{c,p}\Big) + (P_{tr}\times(L+\frac{D_{c,p}}{B}))\Bigg)\times\alpha_{c,p}\Bigg)\right] \tag{2.1}$$

2. ***Minimize Execution Time*** The overall execution time is equal to the time taken by the components running locally and those running remotely.

$$F_2 = min\left[\sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}(1-x_{c,p})(t^l_{c,p}) + \sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}x_{c,p}\Big((t^r_{c,p}+L+\frac{D_{c,p}}{B})\times\alpha_{c,p}\Big)\right] \tag{2.2}$$

3. ***Minimize Memory Consumption*** The total memory consumption in the running personas, is equal to the memory consumed by the components running locally plus the one consumed while waiting the remote execution.

$$F_3 = min\left[\sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}(1-x_{c,p})\times(M^l_{c,p}) + \sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}x_{c,p}(M^r_{c,p}\times\alpha_{c,p})\right] \tag{2.3}$$

4. ***Minimize CPU Usage*** The total CPU usage in the running personas, is equal to the CPU usage of the components running locally plus the one used while waiting for remote execution.

$$F_4 = min\left[\sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}(1-x_{c,p})\times(C^l_{c,p}) + \sum_{p=1}^{m} a_p \sum_{c=1}^{n} b_{c,p}x_{c,p}(C^r_{c,p}\times\alpha_{c,p})\right] \tag{2.4}$$

So our multi-objective optimization problem is: $F = min\{F\_1, F\_2, F\_3, F\_4\}$

S.t

$$\begin{aligned}
&n \in \mathbb{N} && \text{(c1)}\\
&m \in \mathbb{N} && \text{(c2)}\\
&x_{c,p} = \{0,1\} && \text{(c3)}\\
&a_p, b_c, \alpha_{c,p} = \{0,1\} && \text{(c4)}\\
&0 \le M^l_{c,p} \le 1 && \text{(c5)}\\
&0 \le M^r_{c,p} \le 1 && \text{(c6)}\\
&0 \le C^l_{c,p} \le 1 && \text{(c7)}\\
&0 \le C^r_{c,p} \le 1 && \text{(c8)}\\
&F3 \le t_m\% && \text{(c9)}
\end{aligned}$$

$$F4 \leq t_c\% \quad \text{(c10)}$$

Constraints c1 and c2 ensure that the number of personas and their components belong to the set of natural numbers. Constraints c3 and c4 define the binary variables. Constraints c5-c8 ensure that the memory consumption and CPU usage on the mobile device vary between 0 and 1 since they are represented as percentages in our model. Finally, constraints c9 and c10 ensure that the amount of the memory and CPU usages do not exceed certain thresholds based on the capacity of the mobile device. Solving this model will generate the best distribution of the components running in each persona that complies with the formulated objectives aiming to augment personas performance and viability. This distribution is represented by $x_{c,p}$, which represents whether a component c should be offloaded or not. If $x_{c,p} = 0$, then c should run on the mobile device, while it should be offloaded otherwise i.e., for $x_{c,p} = 1$.

2.7 Heuristic Algorithms for Optimal Distribution of Multi-Persona Components

Genetic algorithms (GAs) [Deb (1999)] are heuristic methods that mimic the natural evolution process to solve a problem. Using the concepts of natural selection, GAs simulate the propagation of the fittest individuals over consecutive generations to determine the best solution. Particularly, GAs start by initializing random set of solutions represented by chromosomes/individuals, forming together what is called a population. According to their fitness, solutions from one population are selected to form new candidate solutions called offspring. The fitness of a solution is determined based on a function that aims to minimize or maximize particular objective. The more suitable the solutions are, the more chances they can have to reproduce. To generate offspring, GAs apply crossover and mutation operators to evolve the solutions trying to find better ones. After evaluation, the process is then terminated if stopping criteria is met. Over time, this process will result in increasingly favourable individuals for solving the problem. As such, GAs have been able to prove, through their method of evolution-inspired search, their capability to solve complex optimization problems in many areas [Grefenstette (2013); Wu *et al.* (2014); Cai & Chen (2014)]. In this paper, we exploit the intelligent evolution of solutions in GAs to solve the multi-objective optimization problem of multi-persona. In

what follows we show how the main elements and operators of GAs are mapped to solve the problem.

2.7.1 Representation of Individuals

Each individual is a candidate solution represented as a set of bits having a length of L. Each bit represents a component running in particular persona, and hence the size of an individual is determined based on the number of components. A bit has two possible values 0 and 1. For instance having three components, a randomly generated individual can be represented by 000, 001, 011, 111, 110, 100, 101 or 111. For any component, a bit of 0 is for local execution while a bit of 1 is for remote execution. With this representation, we are able to decode the distribution (local/remote execution) of components running in each persona.

2.7.2 Fitness Evaluation

Genetic algorithms require a fitness function that assigns a score (fitness) to each individual in the current population. The fitness depends on how efficiently that individual can solve the problem at hand. In our multi-persona problem, the fitness of a solution is calculated by evaluating the four objective functions F_1, F_2, F_3, and F_4 that we defined in the previous section. The solutions are ranked based on their ability to minimize these functions.

2.7.3 Operators

2.7.3.0.1 Selection

This operator selects individuals in the population for reproduction. It selects random individuals and picks the x best of them able to minimize mostly the objective functions.

2.7.3.0.2 Crossover

This operator applies modification on individuals selected based on particular rate μ_c to generate offspring. It randomly chooses a bit and exchanges the subsequence before and after that bit between two individuals to create two offspring. For example, the individuals 110 and 101 could be crossed over after the second bit in each to produce the two offspring 111 and 100.

2.7.3.0.3 Mutation

This operator randomly flips some of the bits in individuals selected based on mutation rate μ_m. For example, an individual 010 might be mutated in its second position to yield 000.

2.7.4 Algorithm and Time Complexity Analysis

Algorithm 2.1 MultiPersonaSolver(N, L, μ_m, μ_c)

Input: N := Population size, L := Individual length, μ_m := mutation rate and μ_c := crossover rate
Output: S := Set of fittest individual(s)
Initialize populations index $k := 0$
Generate random population $P_k := GenerateRandomPop(N, L)$
for each individual $i \in P_k$ **do**
    Evaluate objective functions $F_1(i), F_2(i), F_3(i), F_4(i)$
end for
do
    {
    Select x best distribution possibilities and insert them into P_{k+1}
    Crossover $\mu_c \times n$ individuals to produce new offspring
    distributions and insert offspring into P_{k+1}
    Mutate $\mu_m \times n$ individuals by inverting a randomly-selected bit
    in each to generate new possible distribution solutions
    for each $i \in P_{k+1}$ **do**
        Evaluate objective functions $F_1(i), F_2(i), F_3(i), F_4(i)$
    end for
    Increment $k := k + 1$
    }
while stopping criterion is not met
return S the fittest distribution(s) from P_k

Based on these definitions, the algorithm of the solver works as described in Algorithm 2.1. It starts by creating a population of N randomly generated individuals for the running personas.

Each individual is a point in the search space that represents a possible distribution solution. The fitness of an individual is calculated by the computation of the four objective functions F_1, F_2, F_3 and F_4. The fittest individuals in the population are then selected to go through a process of evolution based on crossover and mutation rates μ_c and μ_m respectively. In this process, crossover and mutation operators are applied on the selected individuals to create next generation of individuals for new possible distribution solutions of components. The fitness of these generated individuals is also calculated. This process continues over and over until a stopping criterion is met. This criterion can be number of iterations, time or other relevant condition. Finally, the solver returns the fittest distribution(s) of components to solve the multi-persona problem.

The time complexity of this algorithm depends on many factors, the fitness function evaluation, the population size, the individual length, variation and selection operators and the number of iterations or generations. Initializing and generating the population (Lines 3 and 4 respectively) have time complexity $\mathcal{O}(1)$. The evaluation of the fitness function (Line 5 till Line 7) has time complexity of $\mathcal{O}(N)$ where N is the population size. The tournament selection, single point crossover and bit flip mutation (Line 8 till Line 18), have time complexity of $\mathcal{O}(INL)$ where I is the number of iterations (i.e., generations) and L is the length (i.e., number of bits) of an individual. Finally, the return statement (Line 19) has $\mathcal{O}(1)$. Subsequently, the time complexity of the algorithm is $\mathcal{O}(1) + \mathcal{O}(N) + \mathcal{O}(INL) + \mathcal{O}(1)$ which is equivalent to $\mathcal{O}(INL)$.

2.8 Implementation and Experiments

We dedicate this section to describe the implemented components and discuss our experiments finding.

2.8.1 Implementation

To create personas, we use Cells [Andrus *et al.* (2011)], since by the time this work is done Cells was the first and only open-source virtualization architecture that enables multiple virtual

smartphones and tablets to run simultaneously on the same physical device [Andrus *et al.*]. We set up the environments on Asus Nexus 7 tablet as Cells open source project has been ported for this device only. The tablet runs Android operating system, has quad-core processor and 1 GB of RAM.

On the other hand, and as we discussed throughout the paper, the offloading unit can be an application, service, method or thread. Offloading the entire image of a running application, involves the encapsulation of the latter in a VM instance, which imposes high overhead for creating, cloning, migrating and configuring the VM on the remote server [Shiraz *et al.* (2013); Shiraz & Gani (2014)]. More recent offloading approaches are based on service-level offloading, which distinctly do not have such high overhead and can even reduce the overhead caused by finer granularity components [Shiraz & Gani (2014)]. In addition, Android platform architecture supports and encourages the implementation of applications using activity/service model in android. In this model, the logic code of the computation-intensive tasks is implemented as services through an interface defined using interface definition language (AIDL) [Android] and the user interface as activities. In case the applications do not meet with this requirement, an interface can be easily extracted from the original code as stated by kemp [Kemp (2014)]. Following these facts, a small statistic that we did, in which we downloaded 70 applications from Google play store from different categories (e.g., games, social media, video conferencing, notebook, EMR), showed that 51 % of them contain services varying from 1 to 20 services per app. Therefore, for the sake of the implementation in this paper, we decided to take the services to be the offloadable components using specific libraries [Kemp (2014)].

For the profiler, we implemented one that exploits Linux-based commands to monitor CPU and memory usages. To get the power consumed on idle CPU and screen, active Wi-Fi and during transmission, we use the power profile of android [Android (2017)], whereas, PowerTutor tool [Zhang *et al.* (2010)] is used to get the power consumed on the CPU, screen and network, during the execution of local components. As for the execution time, we implemented and embedded a timer to monitor the execution of the relevant components. The connection between the mobile terminal and the server is done through Wi-Fi network in infrastructure

mode and not in an ad-hoc fashion. So the communication is done indirectly through an access point and not in a peer to peer mode and it is characterized by the IEEE 802.11n standard. Enhancing the implementation of profilers will be part of future work. Finally in the solver, we implemented the decision model based on the metrics that we described in Section 2.6. For the decision maker, we implemented different genetic algorithms in order to compare them and check which one is more adequate in terms of execution time and resource consumption. The first algorithm is NSGA-II [Deb *et al.* (2002)], a multi-objective genetic algorithm which uses pareto ranking mechanism for classification of solutions and crowding distance to define proximity between them. SPEA2 [Zitzler *et al.* (2002)], is another multi-objective evolutionary algorithm based on pareto dominance, yet characterized by its strength scheme that not only takes into account the number of solutions that dominate particular solution, but also the number of solutions by which it is dominated. The third algorithm is SMSEMOA [Emmerich *et al.* (2005)], which is a steady state algorithm, in which a random selection of individuals is done for the mating process and the offspring replaces the individuals of the parent population. Next, IBEA [Zitzler & Künzli (2004)] is an algorithm that employs a quality indicator in the selection process. Finally, MOCell algorithm [Nebro *et al.* (2009)] which is characterized by both decentralized population and archive to store non-dominated solutions. We implemented these algorithms as introduced by their authors, yet based on the mapping that we described in Section 2.7. As for the formulated problem, it can be adapted at any time by modifying the input parameters.

2.8.2 Experiments

The first experiment aims to compare the algorithms described above in order to determine the most efficient one to solve the proposed multi-objective optimization model of multi-persona. While adopting the most performing algorithm, we study in the second experiment the efficiency of our approach compared to two different strategies. In the first strategy, all services in all personas are executed locally on the mobile device, while in the second one, the execution of these services is always offloaded to remote server.

2.8.2.1 Testbed Setup

Based on our results in Section 2.4, running three personas on the mobile device terminal was the most problematic scenario. This makes it the best environment to perform our experiments for both comparing the overhead of the algorithms as well as studying the efficiency of our approach. For fair comparison, we use the same applications that we presented in Section 2.4. Yet as shown in Table 2.4, we vary the number of applications and their distribution in the running personas to reflect different possible usage scenarios of the multi-persona mobile device. To demonstrate the efficiency of our proposition, Table 2.4 includes also S7, which is the scenario that the mobile device was not capable to run, as presented in Section 2.4. S8 is also another scenario that cannot run on the device.

Table 2.4 Distribution of services in different scenarios.

Scenario / **Weight**	S1	S2	S3	S4	S5	S6	S7	S8	S9
Lightweight	3	0	0	1	3	4	6	8	16
Moderate	0	3	0	1	2	3	3	4	0
Heavy	0	0	3	1	1	2	3	3	2
Total	3	3	3	3	6	9	12	15	18

In each scenario, we profiled beforehand all the applications in the running personas as well as the network characteristics in order to generate the input parameters that we presented in Table 2.3. The generated data set forms the input for the decision making algorithm in order to solve the formulated multi-persona problem according to each scenario. The connection is characterized by the IEEE 802.11n wireless networking standard with data rate varying between 54 and 600 Mbps and an average latency of 16 s. The server side is running Ubuntu 12.04 with 7.3GB of memory and quad core AMD Phenom(tm) II X4 B95 processor.

Concerning the algorithms configuration, we used the following values presented in Table 2.5. For NSGA-ii, the population size is 100 individuals and the selection is based on binary tournament. The operators for crossover and mutation are single point crossover and bit flip mutation

Table 2.5 Parameters

Parameter	**Value**
PopulationSize	100 individuals
ArchiveSize	100 individuals
Selection	Binary Tournament
CrossoverProbability	0.9
MutationProbability	1/n (n=number of decision variables)
CrossoverOperator	Single Point
MutationOperator	Bit Flip
CrossoverDistributionIndex	20
MutationDistributionIndex	20
Offset	10
Feedback	20 individuals

with distribution indexes of $\eta_c = 20$ and $\eta_m = 20$ respectively. A crossover probability of $p_c = 0.9$, and a mutation probability of $p_m = 1/n$, where n is the number of decision variables. For SPEA2, both the population and the archive sizes are 100 individuals, and all selection, crossover and mutation operators are the same used in NSGA-ii, with the same values of probabilities and distribution indexes. Also SMSEMOA has the same parameters settings as NSGA-ii, with an offset of 10. Same applies on IBEA and MOCell with feedback of 20 individuals for the latter.

Table 2.6 Number of iterations.

Scenario / **Algo**	S1	S2	S3	S4	S5	S6	S7	S8	S9
NSGA-ii	8	8	8	10	30	60	100	105	135
SPEA2	13	13	13	15	50	100	100	110	135
SMSEMOA	8	8	8	10	50	90	170	170	175
IBEA	14	14	14	15	60	110	-[1]	120	165
MOCell	13	13	13	15	40	60	100	110	135

[1] optimal solution is not found, and therefore IBEA is excluded from being compared with the rest of the algorithms in scenario S7.

The last parameter is the stopping criterion. In these experiments we set it to be the number of iterations needed to find the optimal solution. We set the value of this criterion for each algorithm as illustrated in Table 2.6. The values of this criterion are selected based on multiple executions of these algorithms while trying to find the optimal distribution of services in each scenario. Yet as discussed in Section 2.7, this criterion can be time threshold or any other relevant parameter.

2.8.2.2 Assumptions

- The four objective functions have the same priority level; hence an optimal solution is defined as the best trade-off among them: We experimented other models where we prioritized certain objective functions yet they added no value neither to the algorithms comparison nor to the approach efficiency experiments, therefore we do not present them here.
- Connectivity is always available: Whenever connectivity is not available, our approach does not have to run the solver but rather takes directly the decision of running all the services locally on the device. Therefore, we assume in these experiments that the network is always available, in order to be able to compare the efficiency of the proposed approach whenever offloading the execution of the services is a possible choice.

2.8.2.3 Results and Analysis

In what follows, we present the evaluation of the algorithms to select the best performing one, and the efficiency of our approach compared to other strategies.

2.8.2.3.1 Algorithms Overhead

Figure 2.3 shows the overhead of each algorithm in terms of CPU usage, energy consumption, memory usage and execution time.

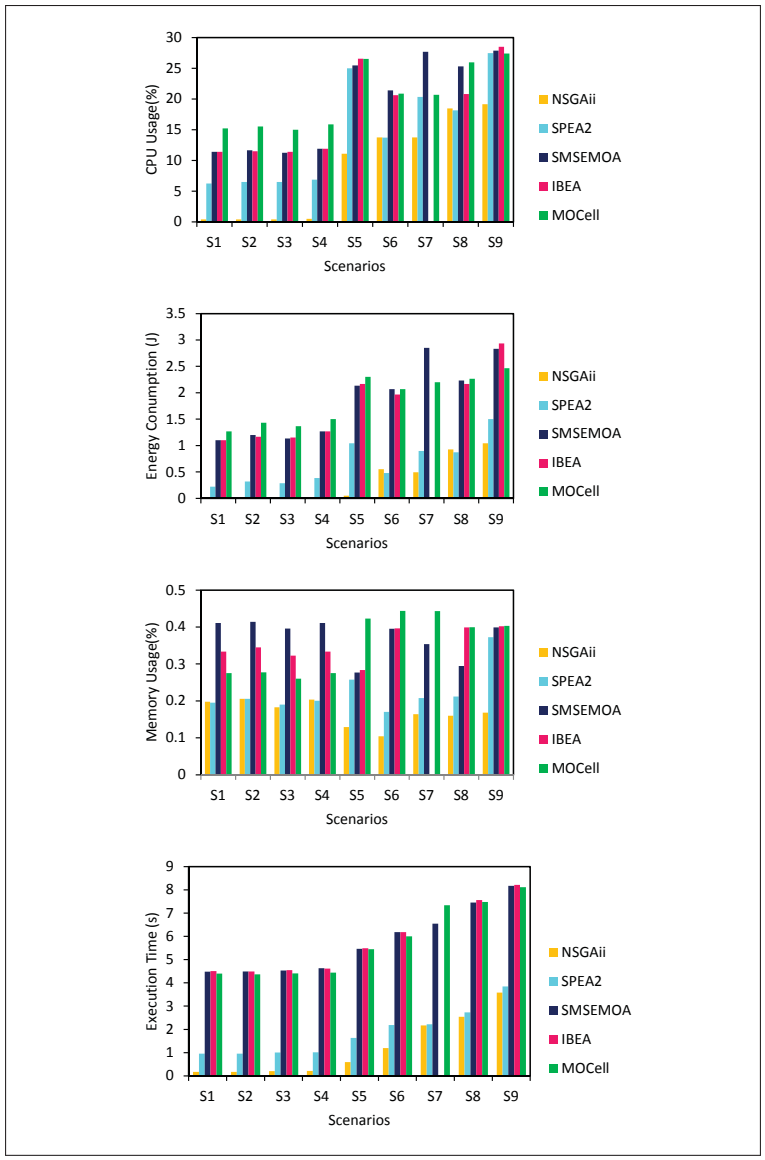


Figure 2.3 Algorithms overhead on the mobile device.

The results show that in most of the scenarios, NSGA-ii algorithm consumed the least CPU compared to the other algorithms. Yet even in scenarios S6 and S8, where SPEA2 algorithm was the best, NSGA-ii had very comparable results. In terms of energy consumption, the results are proportional to those of CPU usage, which can be expected since the energy consumption for the algorithms is the one consumed on their usage of the CPU. Thus, the same analysis applies on the energy consumption results. In terms of memory, NSAG-ii has proved again its efficiency over the other algorithms. The results show that it consumed the least memory in

all the scenarios except S6, where yet it had comparable result to SPEA2, which was the best in this case. Finally, in terms of execution time, NSGA-ii was the fastest to find the optimal solution in all the scenarios. Based on these results, we opted to use NSGA-ii as the decision making algorithm for the solver component of our model.

Table 2.7 Distribution of services based on the decision making algorithm.

Scenario	Decision
S1	000
S2	111
S3	111
S4	101
S5	110111
S6	100110011
S7	100110011001
S8	100110011001100
S9	0000000000000000011

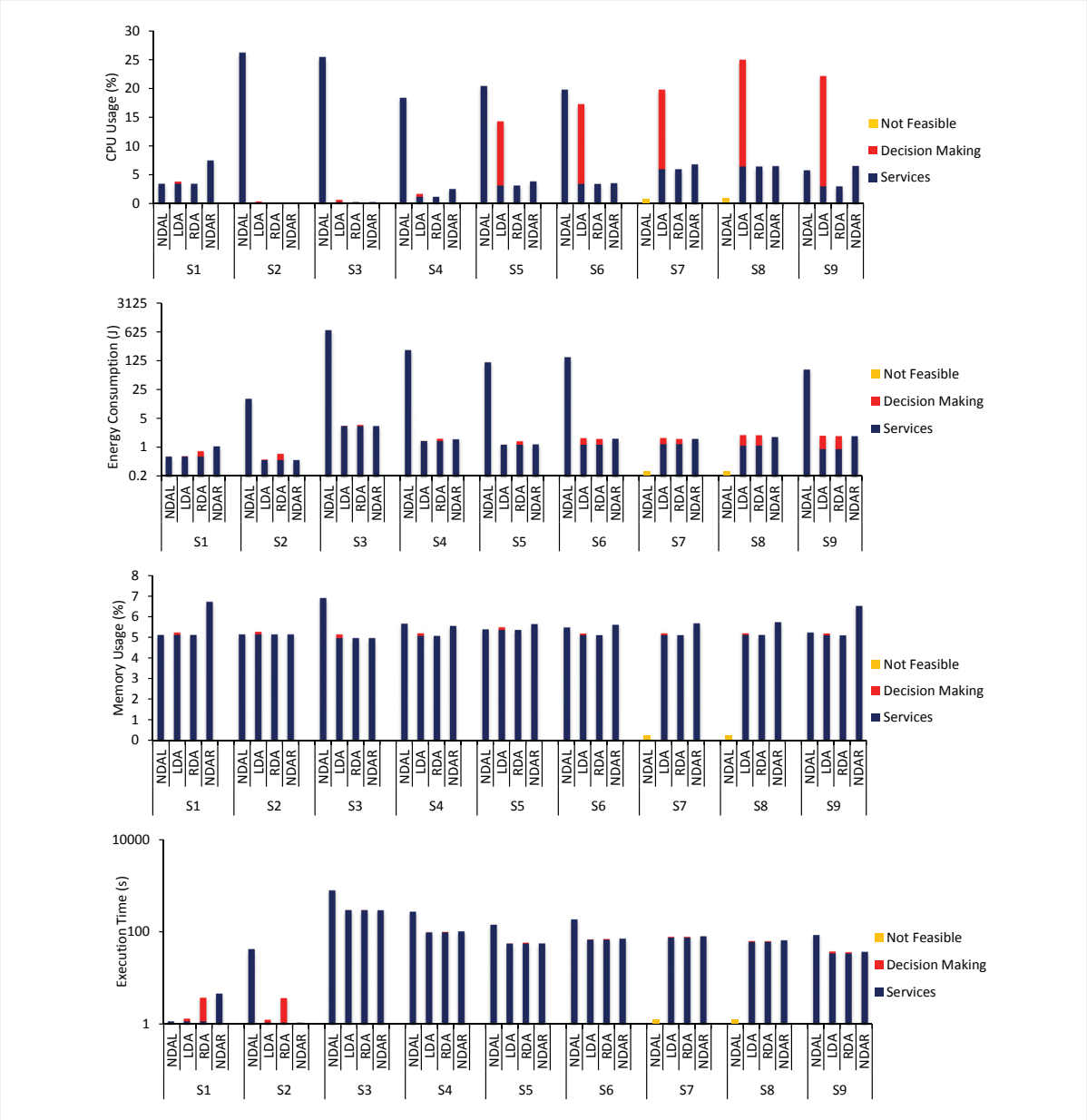
2.8.2.3.2 Approach Efficiency

Table 2.7 shows the optimal distribution of the services running in each scenario based on our solver module that implements NSAG-ii algorithm. The algorithm returns more than just one possible solution in each scenario. Yet, since all solutions are non-dominated, they are considered equally good. Therefore, we can randomly select any of them to be applied. In Table 2.7, we show one of these solutions in each scenario. To recall what we explained in Section 2.8.1, the distribution is represented in a binary set. Each bit corresponds to particular service running in particular persona. A bit having a value of zero means that the decision algorithm recommends to run this service locally, while a bit of one, means that it would be more efficient to offload the service out of the mobile device. For instance in Scenario S4, the virus scanning service was running in the first persona, the unzip service in the second persona, and the NQueens service in the third one. Based on the solver, the optimal distribution that has the best trade-off between the four objective functions in this scenario, is to run the unzip service locally while to offload the two others to be executed on a remote server.

Based on the decision of the solver in each scenario, we study in Figure 2.4 the overhead of our proposition compared to two other different approaches. Current usage of mobile devices involves running all the services in all personas locally on the mobile terminal. Hence, we opted to compare our proposition to such strategy that we call NDAL. In addition, since we are proposing an offloading-based approach to address personas problems, it is indispensable to study whether always running services out of the mobile device can be more efficient. Therefore we compare our proposition to NDAR, where all services are always offloaded to be executed on the remote server. The aim of these experiments is to demonstrate that neither running all the services locally on the mobile terminal nor always offloading their execution to remote server, can offer an efficient multi-persona solution, but rather other appropriate distribution can do. The results of our approach are depicted by LDA and RDA in Figure 2.4. LDA is the case when the solver algorithm is running on the mobile device, while in RDA, it is running on the remote server. Both NDAL and NDAR do not include any decision making process, but rather statically consider local device and remote server, respectively, for the execution of the services. Therefore, the only overhead caused by these approaches is the one of the services, whereas in our approach, we add also the overhead of the solver in the results for reasonable comparison (red bars).

The results in Figure 2.4 show how our approach (LDA and RDA) can remarkably minimize CPU and memory usages, energy consumption and execution time of the services (comparing the blue bars). Particularly, our finding show that, in scenarios S4 throughout S9, our approach was able to achieve way better results, for the four metrics, compared to NDAL and NDAR.

Our approach LDA in scenario S9 is generating higher CPU usage overhead than NDAL due to the solver overload, however we were able to overcome this issue by running the decision making process remotely (RDA) achieving better results than both approaches (NDAL and NDAR) with up to 93% reduction in the CPU usage compared to NDAL in scenario S4. For scenario S1, our approach had results similar to those of NDAL. This is due the optimal distribution found by the solver was to run all the services locally on the device (first row in Table 2.7), which is the same case as of NDAL. Same analysis applies on Scenarios S2 and S3, where



LDA and RDA represent our approach. In LDA, the decision making process is done on the mobile terminal, while in RDA, it is conducted on the server side. NDAL is the case when all components in all personas are running locally on the device, and NDAR is when these components are always offloaded.

Figure 2.4 Approach evaluation.

the optimal distribution found by the solver was to offload all the services (second and third row in Table 2.7), which is the same case as of NDAR. Our approach (RDA) was also able to reduce the memory consumption by 22% compared to NDAR (scenario S9), reach 97%

less energy consumption compared to NDAL and accelerate twice the execution compared to NDAL (scenario S9).

Another interesting observation is in scenarios S7 and S8, where it was impossible to run these scenarios on the device (NDAL yellow bars), but now it is possible using our approach (LDA and RDA) and even with better results than NDAR. The results also show that even after adding the overhead of the decision making process (red bars), either LDA or RDA is still giving better results than those of the other approaches. So for instance, if at any time being the CPU and/or the memory on the multi-persona device goes low, we opt to run the solver on the server side, so it doesn't consume from local CPU and memory. Whereas, in case more priority is given to the energy and/or execution time, the solver will be executed locally since for these metrics LDA had better results than RDA.

Table 2.8 Required number of iterations.

Scenario / **Algo**	S1	S2	S3	S4	S5	S6	S7	S8	S9
NSGA-ii	3	3	3	4	24	30	45	54	67

Finally, in scenarios S1 and S2, the overhead of the decision making process was remarkable. Therefore in what follows, we discuss some alternatives to decrease this overhead. For instance, without running the solver, a decision can be taken based on historical profiled behavior of the same situation. Another interesting idea is to find and generate 'good' solution rather than 'optimal' one, where in this case the algorithm runs for less period and hence decreasing its overhead. Even though on the other hand such proposition increases the overhead of the running services, yet as overall cost it might be beneficial in some scenarios. To investigate this option, we reduced the number of iterations of NSGA-ii in each scenario as depicted in Table 2.8. The results depicted in Figure 2.5 show indeed that in some cases, even though the services have higher overhead but the reduced decision making cost is able to reduce the overall approach overhead. For instances, scenarios S5, S6, S7, S8 and S9 for CPU usage, S5, S6, S8 and S9 in terms of energy consumption, S2, S5 and S7 in terms of memory usage while in S5

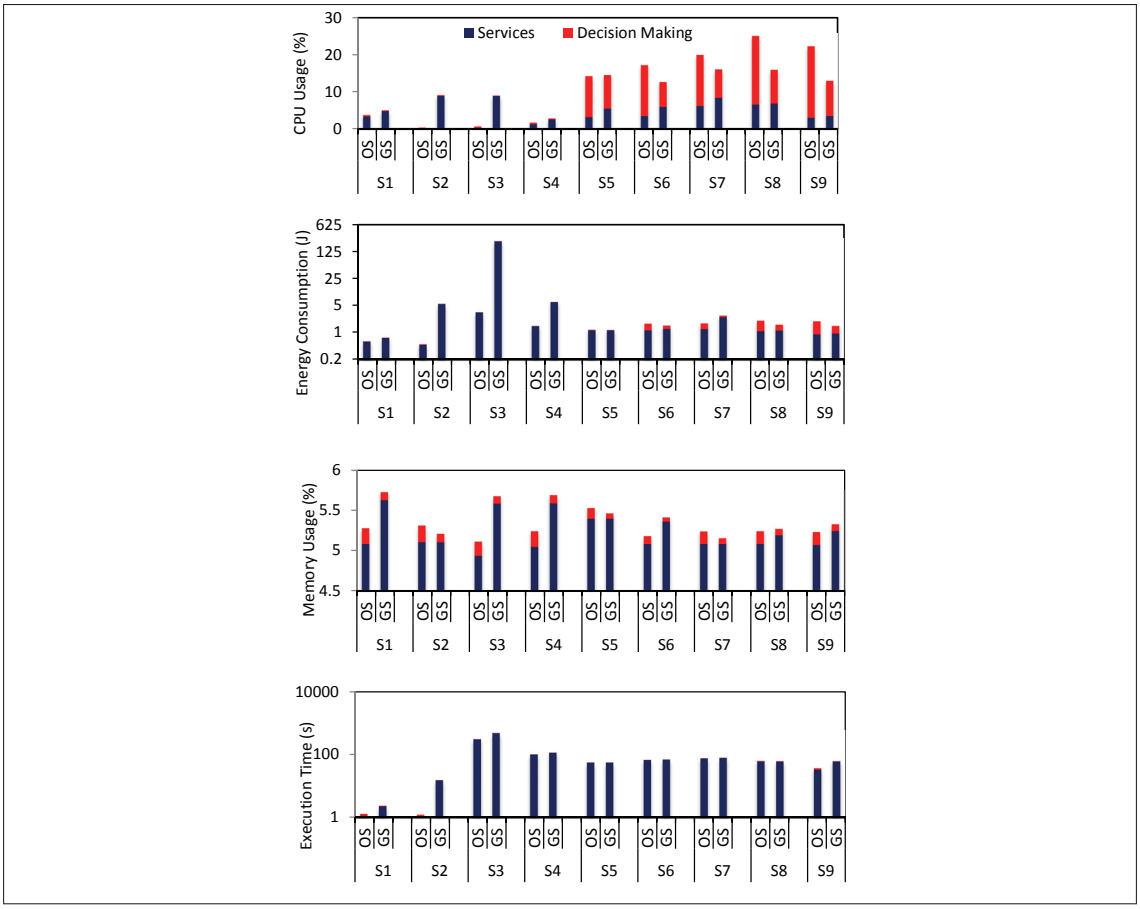


Figure 2.5 Optimal(OS) and Good(GS) solutions overheads.

and S8 for execution time. Yet deeper investigation is still needed and even other alternatives are to be investigated in future work.

2.9 Conclusion and Future Directions

Multi-persona solution is still impeded by the limited resources of mobile devices. These impediments and their implications of putting personas performance and viability on the line have been studied throughout the paper. To address these problems, we presented a novel offloading approach to be integrated with multi-persona on the mobile terminal. Through profiling, multi-objective optimization and heuristics, our proposition is capable of minimizing CPU and memory usages, energy consumption and execution time of the components running in each

persona and hence augmenting the latter performance and viability. Most significantly, it was able to realise scenarios that were not previously feasible to run on the mobile device with multi-persona. Experiments demonstrated the efficiency and qualification of our proposition. Our approach was able in some scenarios to reduce the CPU usage by 93%, the memory usage by 22% and the energy consumption by 97% proved its capability to accelerate the execution of the applications with more than twice faster runtime, compared to existing approaches. This work opens the door for valuable future research directions. Investigating the frequency of calling the profiler on the device is an interesting track, while another valuable direction is to analyze the trade-off achieved between the proposed conflicting optimization objectives [Wu *et al.* (2013); Liu *et al.* (2010); Wu & Wolter (2014); Song *et al.* (2014)].

Acknowledgment

The work has been supported by NSERC Canada and the Associated Research Unit of the National Council for Scientific Research CNRS Lebanon.

CHAPTER 3

ARTICLE 2: SMART MOBILE COMPUTATION OFFLOADING: CENTRALIZED SELECTIVE AND MULTI-OBJECTIVE APPROACH

Hanine Tout[1], Chamseddine Talhi[1], Nadjia Kara[1], Azzam Mourad[2]

[1] Département de Génie Logiciel et TI, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Department of Computer Science and Mathematics, Lebanese American University,
1102 2801 Chouran Beirut, Lebanon

Published in Expert Systems with Applications
Volume 80, Pages 1-13
(DOI: https://doi.org/10.1016/j.eswa.2017.03.011)

3.1 Abstract

Although mobile devices have been considerably upgraded to more powerful terminals, yet their lightness feature still impose intrinsic limitations in their computation capability, storage capacity and battery lifetime. With the ability to release and augment the limited resources of mobile devices, mobile cloud computing has drawn significant research attention allowing computations to be offloaded and executed on remote resourceful infrastructure. Nevertheless, circumstances like mobility, latency, applications execution overload and mobile device state; any can affect the offloading decision, which might dictate local execution for some tasks and remote execution for others. We present in this article a novel system model for computations offloading which goes beyond existing works with smart centralized, selective, and optimized approach. The proposition consists of (1)hotspots selection mechanism to minimize the overhead of the offloading evaluation process yet without jeopardizing the discovery of the optimal processing environment of tasks, (2)a multi-objective optimization model that considers adaptable metrics crucial for minimizing device resource usage and augmenting its performance, and (3)a tailored centralized decision maker that uses genetics to intelligently find the optimal distribution of tasks. The scalability, overhead and performance of the proposed hotspots

selection mechanism and hence its effect on the decision maker and tasks dissemination are evaluated. The results show its ability to notably reduce the evaluation cost while the decision maker was able in turn to maintain optimal dissemination of tasks. The model is also evaluated and the experiments prove its competency over existing models with execution speedup and significant reduction in the CPU usage, memory consumption and energy loss.

Index Terms: Mobile device, Mobile Cloud Computing, Computation offloading, Selective offloading, Hotspots, Optimization.

3.2 Introduction

While smartphone usage continues in a fast-paced mode, developers are provisioning more advanced applications toward all-in-one computing device. However, no matter how sophisticated smartphones are growing, their hardware is still limited in terms of processing power, memory capacity and battery lifetime, compared to their counterparts of desktop machines. With the advancements in wireless communications and the abundance of cloud computing resources, many computations offloading techniques [(Hung *et al.*, 2012; Cuervo *et al.*, 2010; Kosta *et al.*, 2012; Kemp, 2014; Chen *et al.*, 2012; Chun *et al.*, 2011; Shi *et al.*, 2014; Chae *et al.*, 2014; Gordon *et al.*, 2012; Flores *et al.*, 2014; Xia *et al.*, 2014)] have been proposed allowing mobile devices to migrate the execution and throw the burdens of computations to remote infrastructure. Typically, either full application or fine-grained tasks like services, methods, or threads, are migrated to be executed on remote server, releasing the mobile device from intensive processing. The offloading decision is based upon number of factors. The network bandwidth and latency, the resource demands of tasks and the mobile device state all form a context that influences offloading efficiency. According to these aspects, a decision engine analyzes the cost of both local and remote execution and dictates what tasks to be offloaded correspondingly. Though the success of offloading to enhance performance, decrease energy loss and augment resource availabilities, offloading evaluation has its own consequences on the mobile terminal.

Multitasking is a trending action on the mobile operating systems, where multiple applications run simultaneously on the device to meet with the mobile user demands in daily life [(Xiang *et al.*, 2014)]. However, when more than one application are running on the mobile terminal, offloading cannot be evaluated independently for each, as migrating one application or running it locally would affect the execution of the others. Existing offloading techniques necessitate invoking the decision engine for each application separately, which make them unable to efficiently handle such circumstances. In addition, analyzing local and remote execution costs is an iterative process, therefore running such decentralized evaluation imposes in turn significant overhead and forms itself a bottleneck on the end terminal. Although taking the decision remotely might overcome such problem, yet it requires sending relevant data to remote server which imposes more overhead, besides the additional monetary fees to be carried out. Further, existing works assume offloading to be productive whenever remotely executing an application is able to save energy without degrading its normal response time [(Flores *et al.*, 2015)]. However, taking an offloading decision is more complex and additional metrics have to be considered. There is already a lot of understanding in the literature regarding the impact of communication latency and bandwidth, code execution, energy consumption, execution time, CPU and memory in the offloading process, yet existing decision models are limited such that none of them considers all these aspects and tries to reach a tradeoff among them.

This article emphasizes theoretically and experimentally on these limitations and advances relevant prominent solutions. We propose in this work new system model for computations offloading that differs from existing approaches in different aspects and further contributions. This proposition includes first a novel selective mechanism to reduce the search space in offloading evaluation. The mechanism considers only frequently invoked, resource-intensive and time consuming tasks, called hotspots, input for the decision model, in order to reduce the overhead of the decision engine. The work also presents a decision model that considers all of the connectivity properties, energy consumption, CPU and memory usages and execution time of tasks in order to refine the execution environment of tasks. We emphasized in previous work [(Tout *et al.*, 2016)] the effect of such metrics on the device performance and system surviv-

ability. We refine the optimization model in this work to make it resilient not only to the device state but also to the detected hotspots that vary with the device usage, and to strategies that can be enforced on the device through the proposed system to control offloading prioritization and execution suspension of tasks. We also redesign a genetic-based algorithm with tailored adaptive fitness evaluation for intelligent offloading decision making process. The evaluation is centralized to collectively evaluate offloading tasks from different applications running on the mobile terminal. The results of our experiments demonstrate the capability and highlight the efficiency of our proposition. In the following, we use the terms computations, tasks, and components interchangeably.

The originality and novelty of this work are emphasized by the following contributions:

- Novel system model for computations offloading which goes beyond existing techniques with selective, centralized and optimized approach.
- A selective mechanism that minimizes the search space and significantly reduces the overhead of offloading decision evaluation.
- An optimization model with metrics crucial to the device resources and applications performance, adaptable to resource usage, detected hotspots and execution strategies.
- An intelligent centralized decision engine which evaluates the optimization model through tailored genetic-based algorithm able to reach optimal dissemination of tasks with minimal evaluation cost.

The rest of the article is structured as follows. We review in Section 3.3 the basic mobile computation offloading architecture including the role of each component. In Section 4.4, we discuss existing computation offloading strategies while highlighting our contributions. We emphasize on the problems in Section 3.5 and present insights about our approach in Section 3.6. We detail our proposition in Sections 3.7, 4.7 and 4.8. In Section 3.10, we evaluate our proposition and finally in Section 3.11, we conclude the article and draw some future directions.

3.3 Computations Offloading Overview

Mobile cloud computing has integrated cloud computing capabilities into the mobile environment to support mobile devices, ranging from outsourcing software and platforms all the way to infrastructure. In this context, different offloading concepts have been studied. The explosive growth of mobile internet applications, like social networking services, online gaming, audio and video streaming, is the main reason behind the significant overload on the cellular networks. In this context, traffic offloading [(Fiandrino *et al.*, 2015; Han *et al.*, 2010; Andreev *et al.*, 2014)] has been proposed, which is the use of complementary networks like Wi-Fi for data transmission in order to reduce the data carried on the cellular network. On the other hand, the limited resources of mobile devices have triggered another research domain of computation offloading, subject of this work, which has different concept and objective compared to traffic offloading. Code offloading is an opportunistic process that leverages cloud resources (e.g., servers) to execute computation-intensive components designated by a mobile terminal. In this process, an offloading decision is taken based on a cost model that can estimate where the execution is more effective for the end device. Due to mobility and changes in the network conditions, device resources, and computation requirements, the evaluation of this model changes from one execution to another and hence lead to different decisions. Whether the communication between the mobile terminal and the cloud resources is done through the cellular network or Wifi hotspots, the mobile device user is the one responsible for the additional cost imposed by using these channels, which does not raise any economic conflict.

The common architecture of computation offloading is depicted in Figure 3.1. Set of profilers are installed on the terminal to monitor the mobile applications, the environment characteristics, and the device state. The mobile also contains a solver (i.e., decision maker) that based on the information gathered by the profilers, evaluates a cost model and generates an efficient distribution of components (i.e., decides about portions to be executed locally and those to be offloaded). On the other hand, the cloud infrastructure offers the servers where the offloaded components are to be executed. Hereafter, we describe each component of this architecture.

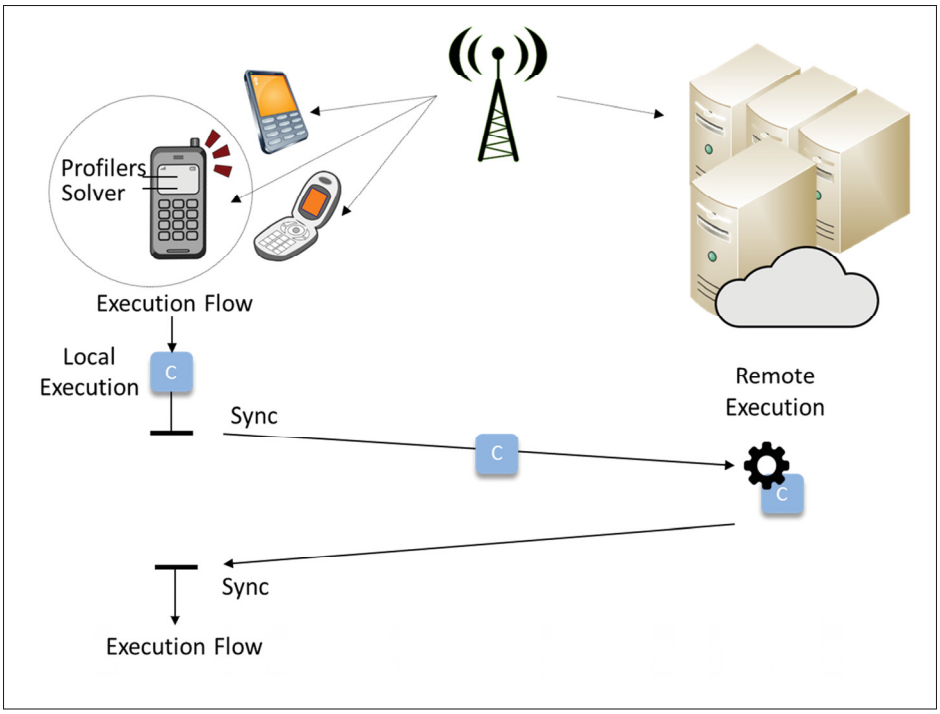


Figure 3.1 Mobile code offloading architecture.

The mobile terminal includes *profilers* to monitor different aspects. The program profiler is responsible of monitoring multiple parameters of the component (C) which is candidate for offloading, like energy consumption, execution time and size of data to be transmitted. The component, which is the offloading unit, can be a service, method or thread inside the application or even a full app. Different methods can be used to identify an offloading candidate, which is also called code partitioning. For instance, some approaches [(Cuervo *et al.*, 2010; Kosta *et al.*, 2012; Kemp, 2014)] rely on the developers to statically annotate explicitly the application source code (e.g., [Remoteable], strategy=remote, @Remote), while others [(Chun *et al.*, 2011)] provide automatic mechanism capable of analyzing the code and generating potential migration points. The network profiler is responsible of monitoring the network characteristics in terms of availability, type (e.g., wifi, 3G), bandwidth, latency and energy consumed on transmission. The device profiler inspects the energy consumption on the device as well as the battery level and CPU utilization to predict critical situations that require offloading, and hence trigger the solver. Based on a cost model, a *solver* evaluates the information gathered

by the profilers. It compares the benefit of local and remote execution and a decision is taken accordingly. If offloading is more beneficial, the code is invoked remotely; otherwise, it is executed locally. The *remote platform* consists of server(s) having higher processing power and more resource competency compared to mobile devices, located in the vicinity like cloudlets [(Satyanarayanan *et al.*, 2009)] or in the cloud [(Amazon, a; Google, a)], which are responsible of executing the offloaded code.

3.4 Related Work

In this section, we review existing offloading approaches and we classify them based on different key factors to distinguish our proposition and highlight its contributions.

Table 3.1 Classification of offloading approaches.

Approach	Offloading Unit	Mobile Cost Model Metrics				Model Evaluation	Evaluation Overhead	Generated Dissemination Savings			
		Energy	Time	Processing	Memory			Energy	Time	Processing	Memory
[(Cuervo *et al.*, 2010)]	Method	✓	x	x	x	Independent	High	O/S	SO/S	SO/S	SO/S
(Chun *et al.*, 2011)	Thread	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
(Gordon *et al.*, 2012)	Multi-Thread	-	-	-	-	-	-	O/US	O/US	O/US	O/US
(Kemp, 2014)	Service	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
(Chen *et al.*, 2012)	Service	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
(Kosta *et al.*, 2012)	Method	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
(Xia *et al.*, 2014)	Method	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
(Chae *et al.*, 2014)	Method	x	✓	x	x	Independent	High	SO/S	O/S	SO/S	SO/S
(Shi *et al.*, 2014)	Method	x	✓	x	x	Independent	High	SO/S	O/S	SO/S	SO/S
(Tout *et al.*, 2016)	Generic	✓	✓	✓	✓	Collective	High	O/US	O/S	O/S	O/S
Our Proposition	Generic	✓	✓	✓	✓	Selective Collective	Low	O/US	O/S	O/S	O/S

[Cuervo *et al.* (2010)] have proposed MAUI, an offloading framework that aims to reduce the energy consumption of mobile applications. The framework consists of a proxy server responsible of communicating the method state, a profiler that can monitor the device, program and network conditions, and a solver that can decide whether to run the method locally or remotely. MAUI uses its optimization framework to decide which method to send for remote execution based on the information gathered by the profiler. The results show the ability of MAUI to minimize the energy consumption of a running app.

CloneCloud is another offloading approach that has been presented by [Chun *et al.* (2011)] in order to minimize the energy consumption and speedup the execution of the running application. A profiler collects the data about the threads running in this app and communicates the gathered data with an optimization solver. Based on cost metrics of execution time and energy, the solver decides about the best partitioning of these threads between local and remote execution. This approach does not require modification in the original application since it works at the binary level. The experiments of CloneCloud showed promising results in terms of minimizing both execution time and energy consumption of an application. However, only one thread at a time can be encapsulated in a VM and migrated for remote execution, which diminishes the concurrency of executing the components of an application.

[Gordon *et al.* (2012)] proposed COMET that rely on distributed shared memory (DSM) systems and virtual machine (VM) synchronization techniques to enable multithreaded offloading and overcomes the limitations of MAUI and CloneCloud, which can offload one method/thread at a time. To manage memory consistency, a field-level granularity is used, reducing the frequency of required communication between the mobile device and the cloud.

Following different strategy, [Kemp (2014)] has proposed Cuckoo that assumes compute intensive code to be implemented as an Android service. The framework includes sensors to decide, at runtime, whether or not to offload particular service since circumstances like network type and status and invocation parameters of the service call on mobile devices get changed continuously, making offloading sometimes beneficial but not always. Cuckoo framework has been able to reduce the energy consumption and increase the speed of computation intensive applications.

[Chen *et al.* (2012)] have proposed a similar framework that automatically offloads heavy backend services of a regular standalone Android application in order to reduce the energy loss and execution time of an application. Based on a decision model, the services are offloaded to an Android virtual machine in the cloud.

ThinkAir has been introduced by [Kosta *et al.* (2012)] as a technique to improve both computational performance and power efficiency of mobile devices by bridging smartphones to the cloud. The proposed architecture consists of a cloud infrastructure, an application server that communicates with applications and executes remote methods, a set of profilers to monitor the device, program, and network conditions, and an execution controller that decides about offloading. ThinkAir applies a method-level code offloading. It parallelizes method execution by invoking multiple virtual machines (VMs) to execute in the cloud in a seamless and on-demand manner achieving greater reduction in execution time and energy consumption.

Considering user delay-tolerance threshold, new offloading-decision making algorithm has been proposed by [Xia *et al.* (2014)]. The proposed tool predicts the average execution time and energy of an application when running locally on the device, then compares them to cloud-based execution cost in order to decide where the application should be executed.

CMcloud is a new scheme that aims to maximize the throughput or minimize the server cost at cloud provider end by running as many mobile applications as possible per server and offer the user's expected acceleration in the mobile application execution. Proposed by [Chae *et al.* (2014)], CMcloud seeks to find the least costly server which has enough remaining resources to finish the execution of the mobile application within a target deadline.

COSMOS system has been presented by [Shi *et al.* (2014)] with the objective of managing cloud resources to reduce their usage monetary fees while maintaining good offloading performance. Through its master component, COSMOS collects periodically information of computation tasks and remote VMs workloads. Based on the gathered information, COSMOS is able to control the number of active VMs over time. Particularly, whenever VMs are overloaded, the system turns on new instance to handle the upcoming requests. It can also decide to shut down unnecessary instances to reduce the monetary cost in case the rest are enough to handle the mobile devices requests.

From Table 3.1, offloading unit identifies the code level granularity where offloading is applied. The mobile cost model metrics represent the decision model aspects used to evaluate offloading

productivity. Model evaluation show the characteristics of the offloading evaluation process, while evaluation overhead reflects the decision making cost. Finally, savings highlight the gain obtained by tasks dissemination generated by the decision engine after cost model evaluation (O and SO stand for optimal and suboptimal while S and US stand for stable and unstable respectively).

3.4.0.3.3 Discussion

Significant attention has been turned toward computation offloading to support mobile devices. Existing approaches focused on enhancing performance and saving energy on the mobile terminal [(Cuervo *et al.*, 2010; Chun *et al.*, 2011; Gordon *et al.*, 2012; Kemp, 2014; Chen *et al.*, 2012; Xia *et al.*, 2014)], others targeted the additional fees imposed by remote execution [(Kosta *et al.*, 2012; Chae *et al.*, 2014; Shi *et al.*, 2014)] and in our turn we focused in previous work [(Tout *et al.*, 2016)] on addressing performance and survivability with multiple virtual environments running on the mobile device. Evaluating the cost model independently for each task or even collectively cause significant overhead and create itself a bottleneck on the mobile terminal with resource constraints, which is a common limitation in all these approaches. Along with this overhead, these approaches fall in suboptimal and unstable savings in the dissemination of tasks due to limited aspects considered in the cost model.

Differently, this paper goes beyond existing works by proposing new system model for computations offloading. As independent offloading evaluation, proposed in existing approaches, fails to handle circumstances when multiple tasks are running simultaneously and hence the decision on a task affects the others, we present a centralized approach able to collectively evaluate offloading tasks from different applications. Also, we propose a selective mechanism to process offloading evaluation only for selected tasks, designated as hotspots, which is capable of significantly reducing the overhead of the decision engine compared to existing approaches. Moreover, the latter evaluates an optimization model that includes energy loss, CPU usage, memory consumption and performance of each component, essential metrics to augment mobile device resources and performance and not being all considered in any of the

existing works. We emphasize the efficiency of these metrics where we prove their ability to overcome suboptimal and unstable savings of tasks dissemination. Differently from exiting models, we also refine the metrics making them resilient not only to the device state but also to the hotspots and to strategies that can be enforced on the mobile terminal like offloading prioritization and suspension of tasks. The decision engine decodes, for the designated components, the execution strategy that achieves a balance between all the metrics considered in the model, reaching stability with high savings in local resource usage and significant execution speedup. Essentially and differently, the proposed computation offloading system model is able to intelligently reduce the overhead of offloading evaluation without jeopardizing optimality in tasks dissemination savings.

3.5 Technical Problems

We highlight in this section the technical problems subject of this work.

3.5.1 Accuracy and Overhead of Decision Model Evaluation

The decision making is an iterative process invoked by the decision engine (solver) and triggered to handle critical situations like processing power degradation, storage inefficiency and dying battery. This process evaluates the decision model to determine whether to offload particular components or not. In existing approaches, the cost model is evaluated for each task independently, which has its own consequences when multiple applications run simultaneously on the mobile terminal to meet with daily life needs. In one hand, with independent offloading evaluation, the system lacks global view of the execution environment which results in inaccurate and faulty decisions. On the other hand, repeating the same process for each task is not reasonable and imposes in turn considerable overhead on the device, higher than the savings achieved by the dissemination if tasks.

The alternative is a centralized collective decision maker that considers the running components from different applications in the evaluation process. Nonetheless, the overhead of

such decision engine increases along with the number of components candidates for offloading. Theoretically, deciding what to offload suffers from an exponential search space in the number of different possibilities in which the components can be distributed (i.e., local or remote execution). It is similar to the various ways n distinct objects (components) can be distributed into m different bins with k_1 objects in the first bin, k_2 in the second one, etc. and $k_1 + k_2 + ... + k_m = n$. By applying the multinomial theorem, this can be calculated as $(k_1 + k_2 + ...k_m)^n = \sum \binom{n}{k_1 k_2 ... k_m} k_1^{n_1} k_2^{n_2} ... k_m^{n_m}$. In our multi-apps offloading problem, $m = 2$, one bin is the mobile terminal and the other is the remote server thus, for n components, there are 2^n different distribution possibilities. This number will dramatically increase with fine grained components like services and will be more crucial with methods and threads as their number is greater inside the applications. Our previous work [(Tout *et al.*, 2016)] suffered also from this dilemma, where the results revealed that the evaluation process in such case can consume up to 6x more CPU usage and 1.25x more energy consumption than the services themselves, and takes up to 3 seconds to find the distribution. These results pushed towards sacrificing optimality of tasks dissemination savings to mitigate the overhead of the evaluation process by accepting suboptimal solution. A possible alternative to overcome such overhead is to take the decision remotely, yet this requires sending relevant data to remote server which not only imposes in turn more overhead, but also additional monetary fees are to be carried out. Therefore the question is how to decrease the overhead of decision making without sacrificing optimal distribution of tasks?

3.5.2 Decision Model Metrics

In the presence of network connectivity and available remote resources, the offloading decision is based upon number of factors. The available resource on the mobile terminal, the latency and bandwidth of the network, the resource demands and performance of the component, any of them can form a context that influences offloading feasibility and efficiency. Besides, energy and execution time, processing power and memory capacity would critically influences the mobile device when running multiple applications simultaneously. As the energy is being

consumed on many elements other than CPU and memory, essentially, graphics, screen and network, decreasing the power consumption does not guarantee more processing power or even more memory availability. As long as the device is on and new applications are running, both CPU and memory usage levels continue to change. Therefore, for example, in case the device is running slow or out of memory, migrating components that require intensive processing and lot of memory would be efficient respectively, regardless of the energy consumption level. Thus, each time an offloading decision is to be made, these factors should be taken into consideration and an explicit evaluation of these metrics should be done. As none of the existing techniques has considered and examined a tradeoff among all these metrics, what is the effect of such multi-objective optimization on the optimality and stability of tasks dissemination savings?, is a question to be answered in this work as well.

3.6 Centralized Selective and Multi-Objective Offloading: Insights

The proposed system model is depicted in Figure 3.2.

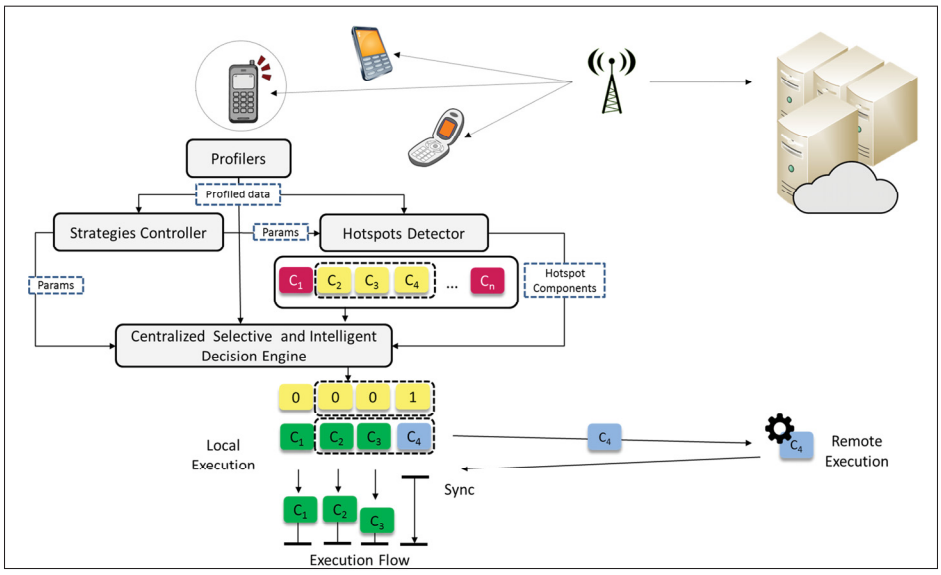


Figure 3.2 System model.

We devote a set profilers on the mobile terminal to monitor the relevant resources. The profilers examine the availability of connectivity, beside its data rate and latency. They also monitor the energy loss, CPU usage, memory consumption , and execution time for all components. A component can be service, method, or even thread that implements certain functionalities. It should be clear that this does not mean that the system deals with components of heterogenous nature but it is generic enough to be applied independently of the offloading unit considered. The gathered profiled data are then communicated with the rest of the modules on the device. Capturing different criteria that influence the resource consumption and performance of the components, the profiled data serve as input for the detector to identify hotspot components forming a subset of the offloading candidates. Frequently invoked, resource-intensive and/or time consuming components are marked as hotspots. Such criteria and the thresholds used for hotspot selection are adjusted by the strategies controller based on the device state, instrumented by the profilers. Gathered data are also used by a centralized, selective and intelligent decision engine, which evaluates a cost model to analyze tasks offloading.

Only selected hotspots (C_2, C_3 and C_4) go into the offloading evaluation process, while the rest of the components (C_1), not included, continue their execution locally on the mobile terminal. The decision model consists of essential metrics that guarantee resource availabilities and enhanced performance on the mobile device. The controller can enforce different strategies in the model to manage offloading evaluation. Such strategies include weights between the model metrics to adapt it with the device state. For example, additional weights can be given to the memory metric when the device is running out of memory. Other strategies include priorities between components like prioritizing offloading from foreground application or even based on criticality classification. Defining such strategies at high level in the controller form part of future track, yet this work considers and adapt to such strategies at the low level of the decision model. The latter is hence adaptable with these controls as well as with the selected hotspots which vary with the device usage. The decision engine includes also a tailored algorithm that examines a tradeoff between the model metrics and dictates accordingly what components to be offloaded and those to run on the mobile terminal (only C_4 is to be offloaded). Offloading

computations can be done through the cellular network of Wifi hotspots. When Wifi capacity is purchased by the cellular network provider, both cellular and Wifi capacities should be considered to evaluate the possibility to process the issued offloading requests as well as to predict the channels ability to handle upcoming requests. Yet, in this work we assume enough resources to handle mobile computations offloading regardless the wireless technology employed (Wifi or cellular network), which is a fair assumption in the light of the resourceful infrastructure of cloud service providers. The following sections detail our proposition.

3.7 Selective Mechanism

To reduce the search space of the decision maker, only hotspot components from the offloading candidates are selected and considered in the evaluation model, while the rest of the components are to be executed locally.

3.7.1 Hotspots Profiling

Profilers on the device generate instrumented data for each and every component. The produced profiles serve as input data used by the detector to identify hot components for selective offloading evaluation. The profilers capture different criteria that influence the resource availabilities and performance on the mobile device. Generated profile includes for every component, the CPU and memory utilization, size and execution time and its frequency of invocation, which are the criteria we use for hotspots selection. The detector modules analyzes the logged profile information and collects all the components whose processing, memory, data size, execution time and frequency of call are greater than given threshold values respectively. These components are marked as hotspots to be considered for offloading evaluation. The higher thresholds will keep the hotter components.

3.7.2 Hotspots Detection

By default, threshold values are set based on bayesian average according to the following formula:

$$\bar{V} = \frac{|C| * m + \sum_{i=1}^{|C|} v_i}{|C| + m} \tag{3.1}$$

where, $|C|$ is the components data set size, m is the prior mean value and v_i is the profile data value. However, to provide efficient decision for the distribution of components, the strategies controller can tune the selection criteria as well as their threshold values according to the mobile device state instrumented by the profilers. The device for instance might suffer from high CPU usage, while after a period of time might run out of memory based on the number and complexity of the applications on the mobile terminal. Thus, the priority of the selection criteria can be tuned according to each situation and some of the criteria might not be considered in the selection of hotspots correspondingly. The thresholds values are also adaptable. For instance, normally, the battery level will clearly drain along with the usage of the device and the execution of applications. Therefore with dying battery, even components with low energy consumption should be considered as hotspot. For this end, the energy threshold value can be reduced by the controller so that even components with small energy cost on the device would be considered. In this article, we refine the decision model to consider these adaptations, yet, at high level, defining such strategies in the controller module is part of future track.

3.7.3 Selection Algorithm

Algorithm 3.1 illustrates the process to select hotspots. Components from different applications, their profiled data, and the thresholds values and a selection ratio to limit the number of components marked as hotspots, all form the input of this algorithm. The process starts by initializing an empty set for the hotspots (Line 3) then loops over all the components calling *ISHOTSPOT* procedure in Algorithm 3.2 for hotspots identification (Line 4 till Line 11). This procedure fills the hotspot set with components having profiled data exceeding the predefined threshold value. Those criteria which are not considered in the selection process have a thresh-

Algorithm 3.1 Hotspots Selection

1: **Input:** Component set $C = \{C_i, ..., C_n\}$, each of which is characterized by size s_i, invocation frequency f_i, CPU usage c_i, Memory usage m_i , Energy consumption e_i, execution time t_i; thresholds for each criteria respectively T_s,T_f,T_c,T_m,T_e,T_t, and selection ratio r
2: **Output:** A subset of hotspot components $H \subseteq C$
3: $H \leftarrow \emptyset$
4: **for** $i = 1$ **to** n **do**
5: ISHOTSPOT(C_i, s_i, T_s, H)
6: ISHOTSPOT(C_i, f_i, T_f, H)
7: ISHOTSPOT(C_i, c_i, T_c, H)
8: ISHOTSPOT(C_i, m_i, T_m, H)
9: ISHOTSPOT(C_i, e_i, T_e, H)
10: ISHOTSPOT(C_i, t_i, T_t, H)
11: **end for**
12: **if** $H = \emptyset$ **then**
13: $H \leftarrow H \cup \{C\}$
14: **end if**
15: **if** $|H| > r$ **then**
16: $NH \leftarrow chooseRand(|H| \times r, H)$
17: $H = NH$
18: **end if**
19: **return** H

Algorithm 3.2 isHotspot

1: **procedure** ISHOTSPOT(C, v, T, H)
2: **if** $T! = -1$ & $C \nexists H$ **then**
3: **if** $v > T$ **then**
4: $H \leftarrow H \cup \{C\}$
5: **end if**
6: **end if**
7: **return** H
8: **end procedure**

old value of -1, as an indicator used by the controller module to vary the selection criteria. *ISHOTSPOT* returns the set of hotspot components back to Algorithm 3.1. If no hotspots were found (Line 12), the set is filled with the initial list of components (Line 13). In case the number of hotspots exceeds a predefined ratio (Line 15), which is by default $n/2$ defined based on empirical study, the number of hot components are limited to that value (Lines 16) with random choice from the components set to avoid selecting the same values. Finally the algorithm returns the hotspots set to be considered in the evaluation model (Line 19).

Lemma 1. The time complexity of Algorithm 3.1 is $O(n)$.

Proof:*The time complexity of the selection algorithm depends solely on the number of the entries n in the components set C. Initializing the empty set H (Line 3) takes* $O(1)$*. Next, running over all the elements in the C set to identify hotspot components (Line 4 to Line 11), takes* $O(n+1)$ *and the inner procedure call is*$Hotspot$ *is of* $O(1)$*. Checking if the hotspot set is empty (Line 12) and subsequently assigning the elements of C to H (Line 13), each takes O(1). Finally, in case the number of elements in H exceeds the ratio r for hotspot components (Line 15), the algorithm will loop again on the elements in H to select the appropriate portion (Line 16), which takes* $O(|H| \times r+1)$*. With* $H \subseteq C$*,* $O(|H| \times r+1)$ *has lower order compared to* $O(n+1)$*, the former can be dropped and thus hotspots can be selected in* $O(n)$*.*

3.8 Centralized Selective Offloading Decision Model

We devote this part to present the proposed centralized, selective and optimized offloading decision model.

Assumptions. We assume in the proposed system model that offloading can be prioritized between components. These priorities are assigned and enforced by the strategies controller. For example, foreground components can have higher priorities to use local/remote resources over background ones. This can be also based on appropriate criticality classification scheme, where more critical apps will be given for example higher priorities to have better performance. Also, the execution of tasks might be suspended based on the device resources. We consider all these strategies at low level in the decision model while their definition at higher level is to be addressed in future work. Additionally, components are assumed to be independent and some of them might not be offloadable like tasks that require local device data. We also consider dynamic environment where the network might not be available and its data rate and latency may vary.

3.8.1 Definition

Definition 1. Considering the set of hotspot components, generated by the selection process, the device state instrumented by the profilers and the management parameters values of the controller, the problem is to find components that should be offloaded and those to be executed locally for a tradeoff of enhanced computing capacity, minimal memory and battery usages on the device and better performance. The problem defined as follows:

Given a set of hotspot components $H = \{c1, c_2, ..., c_k\}$ *on a device D, each of which* c_i *needs* $CPU_{c_i}^{local}$ *processing unit,* $Memory_{c_i}^{local}$ *memory,* $Energy_{c_i}^{local}$ *energy and spends* $ExecTime_{c_i}^{local}$ *period of time when executed on the device; while when offloaded, each consumes* $CPU_{c_i}^{Remote}$ *processing unit,* $Memory_{c_i}^{Remote}$ *memory,* $Energy_{c_i}^{Remote}$ *on the device and needs* $ExecTime_{c_i}^{Remote}$*, waiting and processing the remote response;* α_{c_i} *an indicator whether the component* c_i *is offloadable or not, offloading priorities* p_{c_i}*; network bandwidth Bandwidth and latency Latency, weights* $w_{(F_P)}$*,* $w_{(F_M)}$*,* $w_{(F_E)}$ *and* $w_{(F_T)}$ *for the evaluation metrics; the decision should dictate for each component whether it should be executed locally or remotely in a way to minimize the overall energy loss* F_E*, CPU* F_C *and memory* F_M *usages on the mobile device and speedup the execution* F_T *for better experience.*

Theorem 1. *Offloading optimization decision making is NP-Hard*

Proof:*Offloading optimization can be easily seen in the NP-class; as once a dissemination of components is found, it can be verified in polynomial time. Next, we will prove that this problem is NP-Hard via a reduction from the NP-hard multi-objective-m-dimensional Knapsack Problem (MOMKP) [(Lust & Teghem, 2012)]. We aim in what follows to prove that a solution found for a case of our multi-apps code offloading optimization problem can be used to solve multi-objective-m-dimensional Knapsack. Given the MOMKP - a collection of n items* $a_1, ..., a_n$*, where each item* a_i *has m weights* $w_{ki} \in \mathbb{N}, k = 1, ..., m$ *and t values* $p_{ki} \in \mathbb{N}, k = 1, ..., t$ *and a knapsack of m capacities* $c_k \in \mathbb{N}$*,* $k = 1, ..., m$ *- we can build the offloading optimization decision making problem as follows: Having x applications* $A = \{a_1, ..., a_x\}$ *with y components*

in each, forming a set of $x * y$ representing k components $H = \{c_1, ..., c_k\}$, each corresponding to an item in MOMKP: Set the resource demands of each component c_i in terms of memory and CPU as the weights of the items in the sack. $CPU_{c_i}^{local}$, and $CPU_{c_i}^{remote}$, $Memory_{c_i}^{local}$ and $Memory_{c_i}^{remote}$, are CPU and memory usages when C_i is running locally or executed remotely, respectively. Then, set f_E, f_T, f_P and f_M as the values of each item, where they form the cost of each component in terms of energy consumption, execution time, CPU usage and memory needs respectively. So that $\sum_{i=1}^{x} f_j$ where $j = 1, ..., 4$ formulates each of objective functions in the model correspondingly. Accordingly, the knapsack content is a portion of components selected to run on the mobile device such that the total of each value (cost) is minimized, and vice versa, and a solution to our problem yields a solution to the MOMKP. After this complete proof of the reduction, we conclude that this problem is NP-Hard.

3.8.2 Model Formulation

In the following, we mathematically formulate the proposed offloading evaluation model based on the definitions provided. Built on top of previously proposed optimization metrics in our latest achievement [(Tout *et al.*, 2016)], the model in this work is now resilient not only to the device state but also with the hotspots selection and the execution management strategies defined above.

- Decision Variables:

  $x = \{x_{c_1}, ...x_{c_k}\}$

  where,

  $$\forall c_{i,i:1 \to k}, x_{c_i} = \begin{cases} 0, & \text{if } c_i \text{ is to be executed locally} \\ 1, & \text{if } c_i \text{ is to offloaded} \end{cases}$$

- Parameters:

D	mobile device
H	set of hotspot components

c_i	hotspot component
γ_{c_i}	offloadable component indicator
$ExecutionTime_{c_i}^{local}$	time to execute c_i locally
$ExecutionTime_{c_i}^{remote}$	round trip time to process c_i remotely
$CPU_{c_i}^{local}$	cpu usage on D by c_i executed locally
$CPU_{c_i}^{remote}$	cpu usage on D by c_i offloaded
$Memory_{c_i}^{local}$	memory usage on D by c_i executed locally
$Memory_{c_i}^{remote}$	memory usage on D c_i offloaded
$Data_{c_i}$	size of data transmitted for offloading c_i
$power_{cpu}$	power consumed by D on precessing
$power_{screen}$	power consumed by D on the screen
$power_{idle}$	power consumed by D on idle CPU
$power_{transmission}$	power consumed by D for transmition
p_{c_i}	offloading priority for component c_i
s_{c_i}	suspension indicator for component c_i
$w_{(F_E)}$	weight for function F_E
$w_{(F_T)}$	weight for function F_T
$w_{(F_C)}$	weight for function F_C
$w_{(F_M)}$	weight for function F_M
$\tilde{T}_{F_P}$	threshold for processing load
$\tilde{T}_{F_M}$	threshold for memory consumption
$\tilde{T}_{F_E}$	threshold for energy loss

- Mathematical Model:

$Minimize(F_E, F_T, F_C, F_M)$ where,

$$F_E = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times \left((Power_{cpu} + Power_{screen}) \times ExecTime_{c_i}^{local}\right) + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times \left((Power_{idle} \times ExecTime_{c_i}^{remote}) + \left(Power_{transmission} \times (Latency + \frac{Data_{c_i}}{Bandwidth})\right)\right) \times \gamma_{c_i} \times p_{c_i}\right] \times w_{(F_E)} \quad (3.2)$$

$$F_T = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times (ExecTime_{c_i}^{local}) + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times (ExecTime_{c_i}^{remote} + Latency + \frac{Data_{c_i}}{Bandwidth}) \times \gamma_{c_i} \times p_{c_i}\right] \times w_{(F_T)} \quad (3.3)$$

$$F_C = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times CPU_{c_i}^{local} + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times CPU_{c_i}^{remote} \times \gamma_{c_i} \times p_{c_i}\right] \times w_{(F_C)} \quad (3.4)$$

$$F_M = \left[\sum_{i=1}^{|H|} s_{c_i}(1 - x_{c_i}) \times Memory_{c_i}^{local} + \sum_{i=1}^{|H|} s_{c_i} x_{c_i} \times Memory_{c_i}^{remote} \times \gamma_{c_j} \times p_{c_i} \right] \times w_{(F_M)} \tag{3.5}$$

Subject to

$$\begin{array}{ll} F_P < \tilde{T}_{F_P} & (c_1) \\ F_M < \tilde{T}_{F_M} & (c_2) \\ F_E < \tilde{T}_{F_E} & (c_3) \end{array}$$

The model aims to decrease energy loss, speed up the execution, minimize processing and memory usage on the mobile device, objectives which are defined in Equations (3.2), (3.3), (3.4) and (3.5) respectively with p_{c_i} and s_{c_i} to control prioritization and suspension of tasks accordingly. The model is subject to several constraints; (c_1) forces the decision maker to look for solutions that do not overload processing on the mobile device, (c_2) ensure the candidate solutions do not excess the memory on the end terminal and (c_3) represents the energy constraint that guarantees a threshold for available power on the device.

3.9 Intelligent Decision Making Process

The proposed intelligent decision maker exploits the smart evolution of solutions in genetic algorithms (GAs) [(Deb, 1999)], which have been able to solve complex optimization problems in many areas [(Grefenstette, 2013; Wu *et al.*, 2014; Cai & Chen, 2014)] through their method of evolution inspired search. Based on natural selection, GAs simulate the propagation of the fittest individuals over consecutive generations to determine the best solution. We investigated different algorithms for the decision making process. The first algorithm is NSGA-II [(Deb *et al.*, 2002)], a multi-objective genetic algorithm which uses pareto ranking mechanism for classification of solutions and crowding distance to define proximity between them. Next, SPEA2 [(Zitzler *et al.*, 2002)], which is another multi-objective evolutionary algorithm based on pareto dominance, yet characterized by its strength scheme that not only takes into account the number of solutions that dominate particular solution, but also the number of solutions by

Algorithm 3.3 Intelligent Decision Maker

Input: Set of hotspot components $H = \{c_1, c_2, ...c_k\}$, each of which is characterized by local and remote execution time, cpu usage and memory consumption, network characteristics BL in terms of bandwidth and latency, number of possible solutions N, mutation rate μ_m, crossover rate μ_c and number of generations λ.
Output: Distribution set of hotspots H'.
$i \longleftarrow 0$ ▷ i is the population index
$H' \longleftarrow \emptyset$
$G_i \longleftarrow Random[N][|H|]$ ▷ generates random population
for $k = 1$ to r **do**
    Calculate F_E:=$CalcEnergy(H, G_i, BL)$
    Calculate F_T:=$CalcTime(H, G_i, BL)$
    Calculate F_C:=$CalcProcessing(H, G_i)$
    Calculate F_M:=$CalcMemory(H, G_i)$
end for
for $g = 1$ to λ **do**
    {
    Propagate b best candidates distributions BC for next generation $G_{i+1} \longleftarrow BC$
    select two solutions from G_i, X_A and X_B;
    Generate X_C by evolution-based crossover to X_A, X_B;
    Add X_C to G_{i+2};
    Select a solution X_b from G_{i+2};
    Mutate X_b and generate new feasible solution $X_{j'}$;
    for $k = 1$ to r **do**
        Reexamine F_E:=$CalcEnergy(H, G_{i+1}, BL)$
        Reexamine F_T:=$CalcTime(H, G_{i+1}, BL)$
        Reexamine F_C:=$CalcProcessing(H, G_{i+1})$
        Reexamine F_M:=$CalcMemory(H, G_{i+1})$
    end for
    Update generation $G_i = G_{i+1} + G_{i+2}$
    Update generation index $i \longleftarrow i + 1$
    if Same fitness is detected in G_{i+1} and G_{i+2} **then**
        break;
    end if
    }
end for
$H' \longleftarrow$ optimal components distribution in G_i
return H'

which it is dominated. The third algorithm is SMSEMOA [(Emmerich *et al.*, 2005)], which is a steady state algorithm, in which a random selection of individuals is done for the mating process and the offspring replaces the individuals of the parent population. Further, IBEA [(Zitzler & Künzli, 2004)] is an algorithm that employs a quality indicator in the selection process. Finally, MOCell algorithm [(Nebro *et al.*, 2009)] which is characterized by both de-

centralized population and archive to store non-dominated solutions. We implemented these algorithms as introduced by their authors yet with the adequate mapping. An extensive study presented in previous work [(Tout *et al.*, 2016)] proves the efficiency of nsga-ii [(Deb *et al.*, 2002)] over other algorithms. In this work, we redesign nsga-ii with adaptive fitness evaluation and evolution-based crossover. According to the optimization model that we presented in Section 4.7, the intelligent decision maker is capable of generating the distribution of hot components that minimizes the resource usage and enhance the performance. The process adopted by the decision maker is depicted in Algorithm 3.3. Hereafter, we detail the solution encoding as well as the genetic operators, then in Section 3.10, we study its efficiency.

3.9.1 Solution Encoding

Each chromosome also called individual in a population forms a candidate solution in GAs. For offloading decision optimization, the algorithm starts with population of N randomly generated individuals according to the number of components marked as hotspots (Line 5). Each individual represents the distribution of components. Every individual has a size $|H|$ and it is encoded as a set of binaries $x = \{x_{c_1}, ..., x_{c_k}\}$, where k is the total number of components involved. Each gene x_{c_i} of an individual represents a component on the mobile device and its value dictates whether this component should be executed locally on the end terminal ($x_{c_i} = 0$) or offloaded ($x_{c_i} = 1$).

3.9.2 Fitness Evaluation

In GAs, a score/fitness is designated for each chromosome simulating the propagation of the fittest individuals over consecutive generations in order to determine the best solution. The fitness varies based on how efficiently each individual can solve the problem. In our case, the fitness of each candidate solution is determined by F_E, F_T, F_C and F_M functions that constitute the model proposed in Section 4.7 (Line 6 to Line 11). With all these metrics are to be minimized, the solutions are ranked according to their ability to speedup the execution and reduce the resources usage on the mobile terminal.

The fittest b individuals in the population are then selected to go through a process of evolution (Line 14). In the latter, crossover (Line 15 to Line 17) and mutation (Line 18 to Line 19) operations are applied to produce next generation of individuals for new possible distribution solutions of components. The algorithm reassesses the model metrics to calculate the fitness of these generated individuals (Line 20 to Line 25). The evaluation process continues over and over until any of the stopping criteria is met, where either the fitness of the best individual in successive populations did not improve (Line 28 to Line 30) or the defined number of iterations δ is reached. Finally, the decision maker returns the fittest distribution from the last generation having the optimal tradeoff between the defined metrics (Line 33 to Line 34).

3.9.3 Evolution Process

3.9.3.1 Selection

In this phase, chromosomes are selected to go through the evolution process. We use bit tournament selection, which involves running several rounds over randomly chosen chromosomes from the population. The winner in each round, which has the best fitness is then selected for crossover.

3.9.3.2 Crossover

Crossover is achieved by exchanging genes between two individuals with the intent to produce better offspring. With a μ_c rate, crossover usually occurs when regions of a chromosome break and reconnect to the other chromosome. In contrast, we propose an evolution-based crossover operator. This operator is based on the differential evolution of individuals that optimizes offloading. Taking two parents individuals, genes that produce better fitness (smaller fitness value based on the evaluation presented in Section 4.8.2) when compared to their parents are used to form the offspring.

3.9.3.3 Mutation

Mutation operation is to apply additional modifications in the chromosomes that improve their fitness. With a μ_m rate, we apply standard bit flip mutation.

Lemma 2. The time complexity of Algorithm 3.3 is $O(\lambda N|H|)$

Proof: *The complexity of this algorithm is determined by the fitness function evaluation, the population size, the individual length, variation and selection operators and the number of iterations or generations. Initializing generation index, solution set H' and generating the first random population has each time complexity $\mathcal{O}(1)$. The evaluation of the fitness function has time complexity of $\mathcal{O}(N+1)$ where N is the population size. The tournament selection, evolution-based crossover and bit flip mutation, have time complexity of $\mathcal{O}(N|H|\lambda)$ where $|H|$ is the size of an individual and λ is the number of generations. Reassessing the model is of $\mathcal{O}(\lambda N)$. Finally, the return statement has $\mathcal{O}(1)$. Subsequently, the time complexity of the algorithm is $\mathcal{O}(1)+\mathcal{O}(N+1)+\mathcal{O}(\lambda N|H|)+\mathcal{O}(\lambda N)+\mathcal{O}(1)$. With lower orders are to be dropped, this is equivalent to $\mathcal{O}(\lambda N|H|)$.*

3.10 Numerical Analysis

We devote this section to present the experimental results that demonstrate the efficiency of our proposition.

3.10.1 Testbed Setup

In the following experiments, the mobile terminal is running Android operating system with quad-core processor and 1 GB of RAM. The implementation of a mobile application should follow first particular design pattern in order to make it offloadable. The activity/service model in android allows clear separation between the application code and its user interface. Particularly, the logic code of the computation-intensive tasks can be implemented as services through

an interface definition language (AIDL) while the user interface is defined using activities. Applying such model would facilitate the offloading task as services and activities are already isolated. For this end, we developed three mobile services of different weights that we use in our experiments to map different usage scenarios of the device.

- Zip/Unzip is a lightweight service that allows creating archive folder from files and extracting back the content.
- Virus Scanning is a moderate service that scans files of 100 KB against a library of 1000 viruses' signatures, 1 file at a time.
- NQueens Puzzle is a computation intensive service that implements an algorithm capable of finding all possible solutions of the typical NQueens problem and returns the number of solutions found. In our version we use N=13 to create heavy app.

To make these services offloadable, we take advantage of the offloading libraries provided in [(Kemp, 2014)] as it applies the same design pattern (i.e., activity/service). On first invocation, the offloadable services are sent for remote execution on pre-configured server. Only the .class files of the remote implementation are automatically packaged as jars and transmitted to the server. With just few kilobytes, the transmission of such package drives negligible overhead over the network. Whenever the requested services are already hosted on the server, only relevant parameters are communicated. Yet, the relevant overhead is included in the results. To profile services, we implemented Linux-based commands to monitor CPU and memory usages, while we used PowerTutor [(Zhang *et al.*, 2010)] to monitor the energy consumption on different aspects like CPU, screen and network. As for the power consumption on idle CPU, active network and during transmissions, we took advantage of the power profile in android [(Android, 2017)] to get the relevant information. Finally, we embedded a timer to monitor the execution of each service. As for the decision making algorithm, the configuration is based on empirical study of 50 times of execution. We set $\mu_c = 0.6$, $\mu_c = 1/n$, where n is the number of decision variables, $N = 100$ and the number of generations λ to be 20, 45, 70, 75, 100 for 10 to 50 services respectively. The connection between the mobile terminal and the server is

achieved through WiFi network in infrastructure mode. It is characterized by the IEEE 802.11n wireless networking standard. The server side is running Ubuntu 12.04 with 7.3 GB memory and quad core AMD Phenom(tm) II X4 B95 processor.

3.10.2 Results

3.10.2.1 Decision Model Efficiency

This set of experiments aims to study the efficiency of the proposed offloading optimization model, which considers energy, execution time, CPU and memory metrics in the evaluation process and its effect on the optimality and stability of tasks dissemination savings. Running the developed services, we compare the savings and performance improvement achieved by our proposition to those provided by existing models presented in Table 3.1. The energy model adopted by [Cuervo *et al.* (2010)], the time model adopted by [Shi *et al.* (2014)] and [Chae *et al.* (2014)] and the energy/time model used by [(Kosta *et al.*, 2012; Kemp, 2014; Chen *et al.*, 2012; Chun *et al.*, 2011; Xia *et al.*, 2014)]. Considering the three services described in the setup, we run each of the decision models on the mobile device to make the offloading evaluation and generate the distribution of these services accordingly. Figure 3.3 highlights the savings of the services dissemination found by each model in terms of resource savings and execution speedup.

The energy model shows stable results. However, in terms of optimality, this model can lead to the tasks dissemination that offers the best energy savings yet the minimal in terms of processing, memory and execution speedup. The results show that this model is able to offer 99% less energy with just 33% less CPU usage, 10% less memory consumption and 59% speedup in the execution, with average savings values of 99%, 33%, 10% and 58% respectively. Similarly, the time model shows stability with 485 tasks dissemination that provides the best speedup possible of 63% yet worst in terms of energy with 97%, processing with 33% and memory usage with 10% savings, with average of 63%, 97%, 33%, and 10% savings respectively. On the other hand, the energy/time model is able to reach optimality with respect to processing, mem-

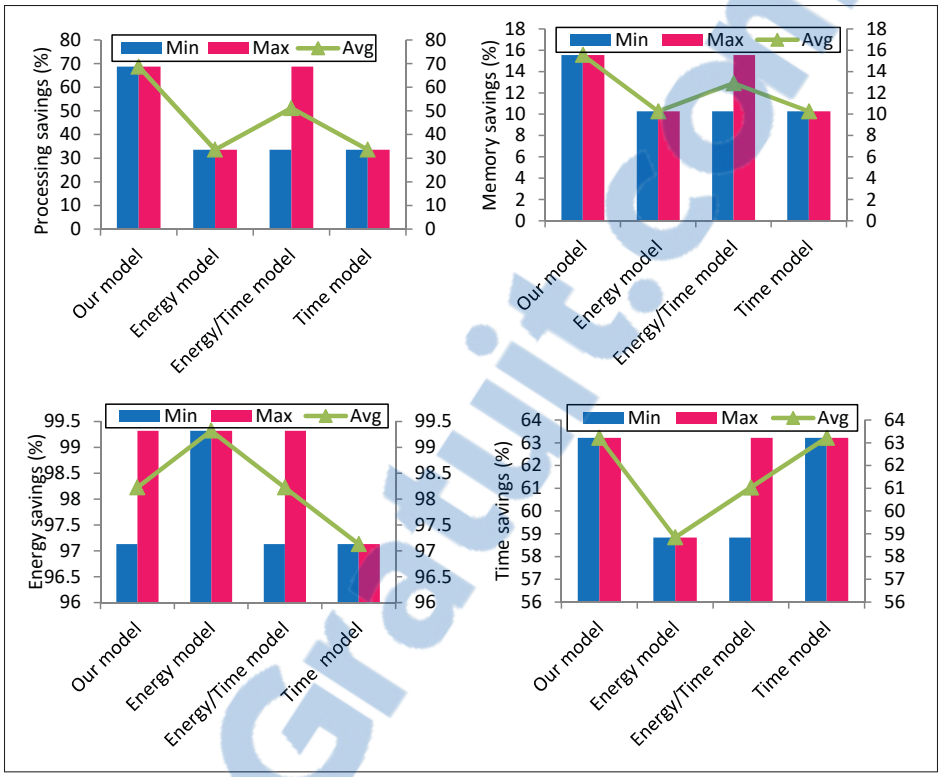


Figure 3.3 Decision savings.

ory, energy and execution's speedup with a dissemination that offers average savings of 51%, 12%, 98% and 61% savings respectively. However, this model shows instability and hence risks finding the dissemination with such values and fall in suboptimal results of 33% 10% 97% and 58% savings accordingly. Per contra, our proposition outperforms the other models and shows stable results in terms of CPU, memory and execution time with 68%, 15.5% and 63% as average reductions respectively. As for the energy, our model can reach up to 99% reduction and 97% in the worst case, with an average of 98%, which is still comparable to the energy/time model. These results confirm that the proposed model is more adequate to offer better trade-off of resources and performance on the device with higher stability.

3.10.2.2 Selective Mechanism and Intelligent Decision Maker Efficiency

The second set of experiments is intended to study the efficiency of the selective method in reducing the overhead of the evaluation process and the ability of the decision maker to adapt

to it without jeopardizing finding the optimal distribution of components. The results of these experiments are depicted in Figures 3.4, 3.5 and 3.6. We increment the number of services stressing the mobile device to study the scalability and cover the case of more fine grained components like methods and threads. We cloned the applications defined in the testbed setup not to implement such large number of services.

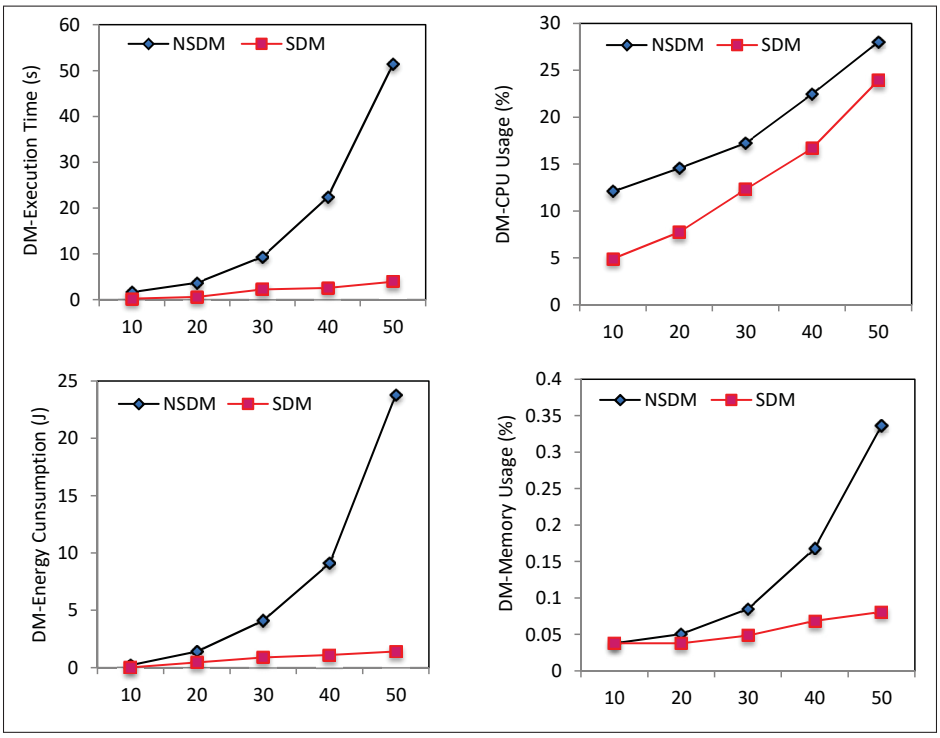


Figure 3.4 Decision maker overhead.

We examine first the overhead of the decision making process (DM) that evaluates our multi-objective optimization model to find the optimal distribution of services (Figure 3.4). The results show that increasing the number of components (i.e., services), imposes significant overhead by the decision maker on the mobile terminal when no selective method is applied (NSDM). Specifically, they show drastic increase in the CPU usage of the decision maker that was 12% with 10 services and reached 28% with 50 services. In addition, its memory usage increased 7 times, its energy consumption increased 114 times and its speed decreased 31 times. Compared to NSDM, our selective method (SDM) was able to reduce 1.2 times the

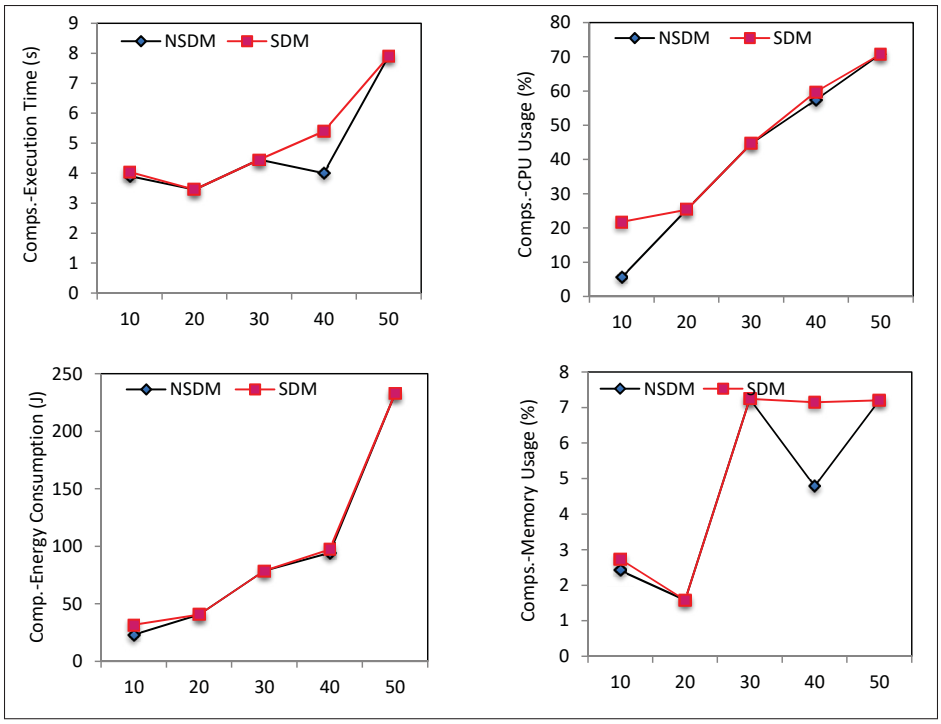


Figure 3.5 Components overhead.

CPU usage of DM, 4 times its memory consumption, 17 times the energy consumption and speedup 13 times its execution.

Following the distribution generated by the decision maker in each scenario, we measured the overhead of the services dissemination as well (Figure 3.5). The objective here is to check how the distribution found by the DM, without considering all the components (i.e., using our selective method SDM) in the cost-benefit analysis, can affect the services overhead. Notably, SDM was able to find the optimal distribution in many scenarios, namely with 20, 30 and 50 services, and hence did not cause any overhead in terms of CPU usage, memory consumption, energy and execution time of the services in these cases.

Another interesting observation can be highlighted when comparing the overall overhead caused by both the DM and the components on the device (Figure 3.6). The results show that even when SDM caused more services overhead than NSDM (scenarios where SDM could not find the optimal solution like when running 40 services), the overall overhead remained better for

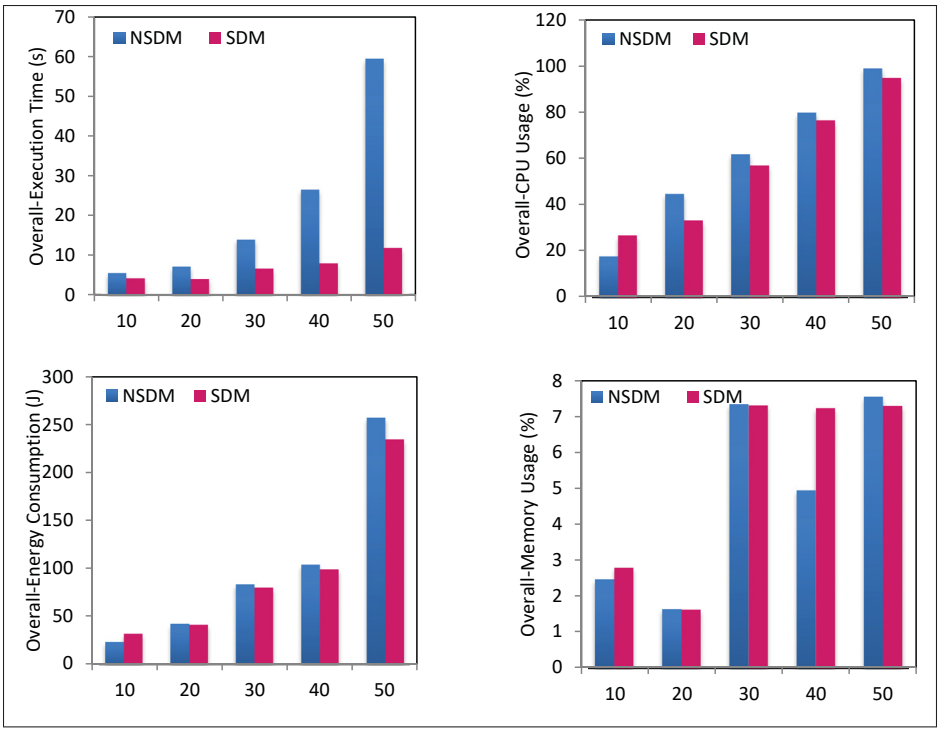


Figure 3.6 Overall overhead.

SDM. This is due to the notable improvement SDM was able to reach in terms of decreasing the overhead of the offloading decision evaluation process.

Finally, we examine the error rates of non-beneficial offloading. The results are illustrated in Table 3.2 The results prove that our proposition is always capable of finding the trade-off

Table 3.2 Decision error rate.

Our model	Energy model	Energy/Time model	Time model
0%	80%	50%	80%

that optimizes the device resources and performance based on the proposed multi-objective optimization model, when other existing models risk finding the adequate computations dissemination.

3.11 Conclusion and Future Directions

This article goes beyond existing approaches with intelligent system model for computations offloading. The system is able to collectively evaluate offloading tasks from different applications that run on the mobile terminal through centralized selective decision engine. With only hotspots considered in the offloading evaluation process, the proposition is capable of significantly reducing the overhead of the decision engine. The latter evaluates a multi-objective optimization model that includes essential metrics to augment mobile device resources and quality of experience. The model is resilient not only to the device state, but also to the detected hotspots and to strategies that can be enforced on the mobile terminal to prioritize offloading and control the execution of tasks. The model was able to offer optimal dissemination of tasks with 68% reduction in the CPU usage, 15.5% in the memory consumption, 99% in the energy and up to 63% execution speedup, with higher stability compared to existing models. According to the model, the decision engine decodes, for the designated hotspots, the execution strategy in order to achieve a tradeoff between the proposed metrics. The selective mechanism was able to notably reduce the overhead of the offloading evaluation process with 1.2 times less CPU, 4x less memory consumption, 17x less energy and 13x speedup while the intelligent decision maker was able to adapt to this mechanism and generate the dissemination of tasks with optimal overall savings.

Promisingly the results give guidance to selective optimized system that can run on resource constrained mobile devices to manage applications executions while alleviating the overhead of the offloading evaluation process without jeopardizing the optimal distribution of tasks that minimizes processing, memory, energy loss and speedup the execution on the device. This work opens several research directions that can be considered by the research community. The main objective is to maintain good quality of experience on the device and ensure longer survivability. Therefore, defining and enforcing management policies and rules between components on the mobile device in order to refine the decision would be a valuable track. While there is still no works that give informative decisions in mobile cloud offloading, adaptive policy-based approaches [(Cimino *et al.*, 2012; Fang *et al.*, 2012)] allow managing situations with awareness

for proactive and instructive recommendations. For instance, rather than dictating what components to offload, more valuable recommendations can be taken based on user preferences, resource availabilities and device state. Such decisions include suspending and shutting down some applications due to resource scarcity and direct decisions to prioritize the mobile device survivability over applications performance. While assuming independent components on the device reduces cost model complexity, considering the dependencies between components of an application is important. With only few works have been proposed in this regard [(Chun *et al.*, 2011; Mahmoodi *et al.*)], dynamic analysis of potential execution flow paths of different tasks in a mobile application has direct impact on the distribution decision where the decision to offload or locally execute particular components can influences the execution of other dependents components.

Acknowledgment

The work has been supported by École de Technologie Supérieure (ETS), NSERC Canada, the Associated Research Unit of the National Council for Scientific Research CNRS Lebanon and the Lebanese American University (LAU).

CHAPTER 4

ARTICLE 3: COST-EFFECTIVE CLOUD-BASED SOLUTION FOR MULTI-PERSONA MOBILE COMPUTING IN WORKPLACE

Hanine Tout[1], Azzam Mourad[2], Nadjia Kara[1], Chamseddine Talhi[1]

[1] Département de Génie Logiciel et TI, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Department of Computer Science and Mathematics, Lebanese American University,
1102 2801 Chouran Beirut, Lebanon

Under Review

4.1 Abstract

Multi-persona mobile computing has begun to make its way to determine the battle about practical strategy for adopting personal devices in workplace. Though its competency, multi-persona performance and viability are critically threatened by the limited resources of mobile devices. Mobile cloud computing (MCC) has risen as promising paradigm that brings cloud benefits to the proximity of mobile terminals, leveraging computations offloading services to address the severity of their resource scarcity. Yet, embracing cloud-based services to augment personas resources and performance raise new concerns. With remote infrastructure being shared between many devices in an institution, the first concern is determining what computations to offload in order to augment multi-persona experience on a broader range of users' terminals. Another concern is the additional remote execution fees imposed on the institution for leveraging offloading services. In this context, we propose new cost-effective solution to address these issues, which includes two-level multi-objective optimization model to settle both concerns and a redesigned genetic algorithm based method for smart and accelerated offloading evaluation. Extensive analysis is performed and the results prove the ability of our proposition to enforce personas by minimizing local processing, memory usage, energy consumption and execution time along with appropriate minimal additional fees.

Index Terms: Mobile device, Multi-persona mobile computing, Mobile Cloud Computing, Offloading, Cost, Optimization, Workplace.

4.2 Introduction

In the age of mobility we are continuously craving new ways to seamlessly automate different aspects of our lives, and mobile devices have become one of the most eminent ways to do so. Yet, the rapid development in consumer electronics has put some pressure on organizations. BYOD (Bring your own device) [Rouse (2012a)] trend has been embraced across a variety of businesses as a way to welcome changes in traditional work models. However, a battle has been opened over the strategy that would allow the inclusion of personal devices in the work environment without threatening the users' privacy and imposing risks on professional data. Recent technological advancements have paved the way for multi-persona mobile computing to become the most seamless and efficient candidate to determine this battle. Multi-persona allows having separate personas on a single device untangling concerns resulting from mixing personal and professional realms [Eiferman (2014a)]. This technology is being welcomed in different areas. In enterprises, employees can manage wallet, games, social media and business personas, all on the same mobile device with security policies of different levels enforced on each. Likewise in healthcare, multi-persona delivers on refining the way of practicing medicine [NADLER (2014)]. The patterns of delivering care for patients have changed. Doctors are not tied anymore to their private clinics but rather work additionally with multiple institutions, where they are subject to different policies. Besides the work/life balance multi-persona is able to offer, apps are segregated into various personas in isolated virtual environments. Therefore, through different personas for clinic and hospitals practices, multi-persona mobile computing allow doctors to fit among their multiple places of work, by complying with the policy of each, and the quality of care they provide to their patients.

Despite the evolution of mobile devices, a solid emergence of multi-persona is still impeded by the resource constraints of such platform. In a recent study [Tout *et al.* (2015)], we revealed the inability of the mobile device resources to afford high performing personas or tolerate their

viability. Yet, in this mobile world that we live in, considerable effort has been put forward in MCC, offering new services for mobile devices to overcome their resource limitations [Zhou *et al.* (2015); Khan *et al.* (2014); Abolfazli *et al.* (2014); Deng *et al.* (2015); Chen (2015); Zhang *et al.* (2015)]. Several offloading techniques [Zhou *et al.* (2016); Hung *et al.* (2012); Cuervo *et al.* (2010); Kosta *et al.* (2012); Kemp (2014); Chen *et al.* (2012); Chun *et al.* (2011); Shi *et al.* (2014); Chae *et al.* (2014); Gordon *et al.* (2012); Flores *et al.* (2014); Xia *et al.* (2014)] have been proposed allowing mobile terminals to migrate the execution of applications to resourceful infrastructure. Typically, components like services, methods, or threads, inside an application, are migrated to be executed on remote server, releasing the mobile device from intensive processing. Each of these works has proposed an offloading evaluation model to meet with different objectives like minimizing energy consumption or accelerating applications execution. A decision is taken accordingly dictating where the components should be executed whether locally or offloaded to meet with such objectives. These techniques proved their ability to enhance the performance of the applications and extend the battery lifetime of a mobile device.

We, in our turn, were able to augment the experience of multi-persona, where we proposed in previous work [Tout *et al.* (2016)] an offloading-based solution to augment personas performance and viability on mobile terminals. The solution was capable of significantly reducing the CPU usage, memory consumption, and energy loss and proved its ability to accelerate the execution of the applications. The results also demonstrated its ability to tolerate different scenarios that cause shutting down personas on the mobile device.

However, to embrace such solutions in workplace where multi-persona mobile computing is primarily proposed, two other crucial challenges arise. In workplace, many multi-persona devices are involved in the business model sharing the remote cloud resources. As a case in point, in the healthcare system, these devices are used by a variety of medical practitioners with applications that allow them to access medical records to provide care to their patients no matter their place of work. While sharing remote assets, the first concern now is to adequately disseminate the execution of services between local and remote processing to enforce as many

personas as possible on all users' terminals engaged rather than a solo device. Additionally, leveraging commercial cloud resources is not free of charge. With several offloading requests generated from numerous mobile devices, the fees aspect becomes a key factor in the equation. Thus, the second concern that arises is the need to reduce the remote execution fees, which form additional expenses for the institution itself. Therefore, the challenging question that this paper aims to answer is how to optimize the total cost in such model? Particularly, how to balance in one hand, the usage of cloud-based offloading services to minimize processing, memory, energy and execution time in personas on as many devices as possible, and on the other hand the remote execution fees imposed?

To the best of our knowledge, this work is the first to address this problem. In this new proposition, a comprehensive decision is clearly competent to meet with both fundamental concerns of different perspectives, rather than a decentralized offloading decision taken separately on each multi-persona device as the latter lacks overall view of the system. For this end, we devote a centralized smart decision maker that instructs each terminal on the computations execution strategies that achieve the optimal cost. Differently from existing works including our previous one [Tout *et al.* (2016)], our proposition applies multi-layer optimization encompassing different extents. To evaluate the offloading service, we propose a novel two-level optimization model that not only considers metrics of CPU, memory, energy, and execution time but also personas on each and every device as well as the corresponding overall remote execution fees imposed. Additionally, to solve this model, we tailor a genetic-based optimization algorithm whose solution disseminates computations between local and remote execution to achieve the needed cost balance. The algorithm presents new additional evolution components tailored to optimize the model and speed-up the decision taking process. We define the cost as resource usage and performance on every multi-persona device in addition to their total usage fees of remote resources. While clearly a centralized approach should yield optimal trade-off for both concerns, we also study how such solution is far from suboptimal ones obtained by the decentralized distributed paradigm and we show how both approaches can be efficient from different perspectives. In the following, we use the terms computations, components and services inter-

changeably. The following list of contributions emphasize the novelty and originality of our proposition:

- Novel scheme for efficient cloud-based support of multi-persona in workplace.
- Two-level cost-effective optimization model to balance processing, memory, energy and performance of personas on different devices with minimal remote resources usage fees.
- A centralized smart and cost-effective decision algorithm to generate the dissemination of tasks that provides optimal cost. The algorithm presents with new additional evolution components tailored to optimize the model and speed-up the decision taking process.
- Evaluation of both centralized and decentralized approaches and verifying their engagement to address the raised concerns from different perspectives.

The rest of the paper is structured as follows. We give in Section 4.3 an overview of mobile computation offloading and we discuss its advantages to support mobile terminals. In Section 4.4, we review existing strategies while highlighting some technical aspects. We present an illustrative business model and emphasize on the problems in Section 4.5. We outline our proposition in Section 4.6, and present its core details in Sections 4.7 and 4.8. In Section 4.9, we evaluate our approach and provide list of learned lessons. Finally in Section 4.10, we conclude the article and draw some future directions.

4.3 Computation offloading to support mobile devices: Background

We review in this section the main concepts in computation offloading. The latter is an opportunistic process that leverages cloud resources (e.g., servers) to execute computation-intensive components designated by a mobile terminal. In this process, an offloading decision is taken based on a cost model that can estimate where the execution is more effective for the end device. Due to mobility and changes in the network conditions and device resources, the evaluation of this model changes from one execution to another and hence lead to different decisions.

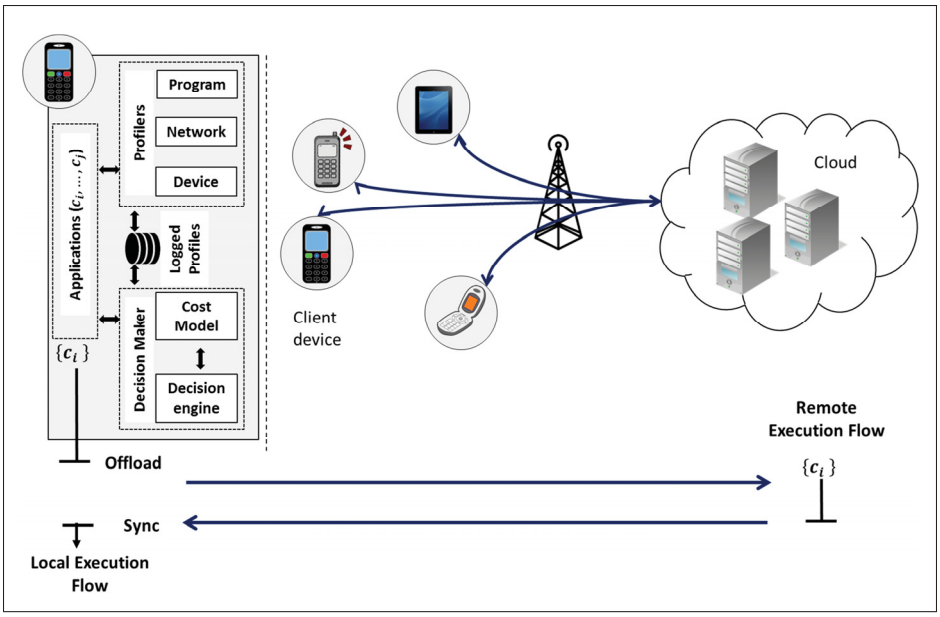


Figure 4.1 Mobile computation offloading.

The common architecture of computation offloading is depicted in Figure 4.1. Set of profilers are installed on the terminal to monitor the mobile applications, the environment characteristics, and the device state. The mobile also contains a solver that based on the information gathered by the profilers, evaluates a cost model and generates an efficient distribution of components (i.e., decides about portions of applications to be executed locally and those to be offloaded). On the other hand, the cloud infrastructure offers the servers where the offloaded components are to be executed. Hereafter, we describe each component of this architecture.

Profilers: The mobile terminal includes profilers to monitor different aspects. The program profiler is responsible of monitoring multiple parameters of the component c_i, which is candidate for offloading, like energy consumption, execution time and size of data to be transmitted. The component can be a service, method or thread inside the app. Different methods can be used to identify an offloading candidate. For instance, some approaches [Cuervo *et al.* (2010); Kosta *et al.* (2012)] and [Kemp (2014)] rely on the developers to annotate explicitly the application source code (e.g., [Remoteable], strategy=remote, @Remote), while others [Chun *et al.* (2011)] provide automatic mechanism capable of analyzing the code and generating potential migration points. The network profiler is responsible of monitoring the network characteris-

tics in terms of availability, type (e.g., wifi, 3G), bandwidth, latency and energy consumed on transmission. The device profiler inspects the energy consumption on the device as well as the battery level and CPU utilization to detect critical situations that require offloading, and hence trigger the solver.

Solver: Based on a cost model, the decision maker evaluates the information gathered by the profilers. The evaluation model explores a trade-off between different metrics like energy consumption and execution time of the applications. In the evaluation process, the solver compares the benefit of local and remote execution and a decision is taken accordingly. If offloading is more beneficial, the code is invoked remotely; otherwise, it is executed locally.

The *remote platform* consists of server(s) located in the vicinity like cloudlets [Satyanarayanan *et al.* (2009)] or in the cloud [Amazon (a); Google (a)], which are responsible of executing the offloaded code. Having higher processing power and more resource competency compared to mobile devices, these servers are able to accelerate the execution time and augment the user experience on the mobile terminal.

4.4 Related Work

We survey in this section the most common and recent offloading techniques, then we classify them based on different key factors.

Table 4.1 Taxonomy of offloading schemes.

Scheme	Type	Target			Offloading Unit	Decision Metrics					Cost Evaluation	Evaluation Overhead	Savings				
		Application	Persona	Device		Energy	Time	Processing	Memory	Fees			Energy	Time	Processing	Memory	Fees
mCloud[Zhou *et al.* (2016)]	MC	Single	Single	Single	Method	✓	✓	x	x	x	Independent	High	+++	+++	+	+	-
MAUI[Cuervo *et al.* (2010)]	MC	Single	Single	Single	Method	✓	x	x	x	x	Independent	High	+++	+++	+	+	-
CloneCloud[Chun *et al.* (2011)]	MC	Single	Single	Single	Thread	✓	✓	x	x	x	Independent	High	+++	+++	+	+	-
COMET[Gordon *et al.* (2012)]	MC	Single	Single	Single	Multi-Thread	-	-	-	-	-	-	-	+++	+++	+	+	-
Cuckoo[Kemp (2014)]	MC	Single	Single	Single	Service	✓	✓	x	x	x	Independent	High	+++	+++	+	+	-
Chen et al.[Chen *et al.* (2012)]	MC	Single	Single	Single	Service	✓	✓	x	x	x	Independent	High	+++	+++	+	+	-
Phone2Cloud[Xia *et al.* (2014)]	MC	Single	Single	Single	Method	✓	✓	x	x	x	Independent	High	+++	+++	+	+	-
ThinkAir[Kosta *et al.* (2012)]	CC	Single	Single	Single	Method	✓	✓	x	x	✓	Independent	High	+++	+++	+	+	+++
COSMOS[Shi *et al.* (2014)]	CC	Multiple	Single	Multiple	Method	x	✓	x	x	✓	Collective	Low	+	+++	+	+	+++
CMcloud[Chae *et al.* (2014)]	CC	Multiple	Single	Multiple	Method	x	✓	x	x	✓	Collective	Low	+	+++	+	+	+++
Tout et al.[Tout *et al.* (2016)]	MC	Multiple	Multiple	Single	Generic	✓	✓	✓	✓	x	Collective	High	+++	+++	+++	+++	-
Our Proposition	**MC**	**Multiple**	**Multiple**	**Multiple**	**Generic**	✓	✓	✓	✓	✓	**Collective**	**Low**	**+++**	**+++**	**+++**	**+++**	**+++**

Table 4.2 Taxonomy of offloading schemes.

Scheme	Proactivity	Management Strategies
mCloud[Zhou *et al.* (2016)]	No	Offload
MAUI[Cuervo *et al.* (2010)]	No	Offload
CloneCloud[Chun *et al.* (2011)]	No	Offload
COMET[Gordon *et al.* (2012)]	No	Offload
Cuckoo[Kemp (2014)]	No	Offload
Chen et al.[Chen *et al.* (2012)]	No	Offload
Phone2Cloud[Xia *et al.* (2014)]	No	Offload
ThinkAir[Kosta *et al.* (2012)]	No	Offload Cloud Resource Manamegement
COSMOS[Shi *et al.* (2014)]	No	Offload Cloud Resource Manamegement
CMcloud[Chae *et al.* (2014)]	No	Offload
Our Proposition	**Yes**	**Offload** **Turn off Component** **Switch off Persona**

4.4.1 Mobile Centric Offloading

Several approaches have proposed mobile centric offloading schemes that proved their ability to enhance the applications performance and minimize the energy consumption on the mobile device.

In mCloud framework [Zhou *et al.* (2016)], different cloud resources are considered; mobile ad-hoc device cloud, cloudlets and public cloud. The work aims to find where tasks should be executed so that the overall energy consumption and execution time is the lowest among all cloud resources in the mobile cloud infrastructure based on the current state of the device.

MAUI [Cuervo *et al.* (2010)] is an offloading framework that has been proposed by Cuervo et al. in order to reduce the energy consumption of mobile applications. The framework consists of a proxy server responsible of communicating the method state, a profiler that can monitor the device, program and network conditions, and a solver that can decide whether to run the method locally or remotely. MAUI uses its optimization framework to decide which method to

send for remote execution based on the information gathered by the profiler. The results show the ability of MAUI to minimize the energy consumption of a running app.

CloneCloud [Chun *et al.* (2011)] is another offloading approach that has been presented in order to minimize the energy consumption and speed-up the execution of the running application. A profiler collects the data about the threads running in this app and communicates the gathered data with an optimization solver. Based on cost metrics of execution time and energy, the solver decides about the best partitioning of these threads between local and remote execution. This approach does not require modification in the original application since it works at the binary level. The experiments of CloneCloud showed promising results in terms of minimizing both execution time and energy consumption of an application. However, only one thread at a time can be encapsulated in a VM and migrated for remote execution, which diminishes the concurrency of executing the components of an application.

Relying on distributed shared memory (DSM) systems and virtual machine (VM) synchronization techniques, COMET [Gordon *et al.* (2012)] enable multithreaded offloading and overcomes the limitations of MAUI and CloneCloud, which can offload one method/thread at a time. To manage memory consistency, a field-level granularity is used, reducing the frequency of required communication between the mobile device and the cloud.

Kemp has followed different strategy and proposed Cuckoo [Kemp (2014)] that assumes compute intensive code to be implemented as an Android service. The framework includes sensors to decide, at runtime, whether or not to offload particular service since circumstances like network type and status and invocation parameters of the service call on mobile devices get changed continuously, making offloading sometimes beneficial but not always. Cuckoo framework has been able to reduce the energy consumption and increase the speed of computation intensive applications.

Chen et al. [Chen *et al.* (2012)] have proposed a similar framework that automatically offloads heavy back-end services of a regular standalone Android application in order to reduce the

energy loss and execution time of an application. Based on a decision model, the services are offloaded to an Android virtual machine in the cloud.

An offloading-decision making algorithm that considers user delay-tolerance threshold has been proposed by Xia et al. [Xia *et al.* (2014)]. The tool predicts the average execution time and energy of an application when running locally on the device, then compares them to cloud-based execution cost in order to decide where the application should be executed.

4.4.2 Cloud Centric Offloading

Other schemes have proposed cloud centric solutions that focus on how to manage cloud resources in order to reduce the remote execution fees, while maintaining good performance on the mobile terminal.

ThinkAir [Kosta *et al.* (2012)] has been introduced as a technique to improve both computational performance and power efficiency of mobile devices by bridging smartphones to the cloud. The proposed architecture consists of a cloud infrastructure, an application server that communicates with applications and executes remote methods, a set of profilers to monitor the device, program, and network conditions, and an execution controller that decides about offloading. ThinkAir applies a method-level code offloading. It parallelizes method execution by invoking multiple virtual machines (VMs) to execute in the cloud in a seamless and on-demand manner achieving greater reduction in execution time and energy consumption.

Shi et al. have presented COSMOS system [Shi *et al.* (2014)] with the objective of managing cloud resources to reduce their usage monetary cost while maintaining good offloading performance. Through its master component, COSMOS collects periodically information of computation tasks and remote VMs workloads. Based on the gathered information, COSMOS is able to control the number of active VMs over time. Particularly, whenever VMs are overloaded, the system turns on new instance to handle the upcoming requests. It can also decide to shut down unnecessary instances to reduce the monetary cost in case the rest are enough to handle the mobile devices requests.

Chae et al. [Chae *et al.* (2014)] have proposed CMcloud, a new scheme that aims to maximize the throughput or minimize the server cost at cloud provider end by running as many mobile applications as possible per server and offer the user's expected acceleration in the mobile application execution. CMcloud seeks to find the least costly server which has enough remaining resources to finish the execution of the mobile application within a target deadline.

4.4.3 Analysis

Table 4.2 presents a taxonomy of most relevant works to clearly differentiate between them. Type represents the nature of each scheme whether it proposes mobile or cloud centric management solution (MC and CC respectively). Target represents the number of applications, personas and devices considered in each technique. Offloading unit identifies the code level granularity where offloading is applied. The decision metrics represent the aspects to be evaluated in order to determine the offloading productivity. Model evaluation show the characteristics of the offloading evaluation process, while evaluation overhead reflects the decision making cost. Finally, savings highlight the gain obtained in terms of resource usage preservation and performance acceleration on the mobile terminal in addition to the monetary fees reduction for cloud offloaded services (+++ represents higher savings compared to +, while - for not applicable).

Several researchers have proposed computation offloading schemes to support mobile devices. These approaches have indeed proved their abilities to enhance the performance and save energy on the mobile terminal [Zhou *et al.* (2016); Hung *et al.* (2012); Cuervo *et al.* (2010); Kosta *et al.* (2012); Kemp (2014); Chen *et al.* (2012); Chun *et al.* (2011); Shi *et al.* (2014); Chae *et al.* (2014); Gordon *et al.* (2012); Flores *et al.* (2014); Xia *et al.* (2014)]. In our turn, we proposed in previous work an offloading-based solution, yet differently for multi-persona devices, which was able to augment personas performance and viability [Tout *et al.* (2016)]. The work addressed the needs of multi-persona on a single mobile device by analyzing services offloading opportunities that optimize the resource usage and performance on the end terminal.

Yet, this paper goes beyond existing works, including our previous contribution, by addressing the concerns of embracing such solutions in workplace, where in this new architecture, resources and performance of personas on various devices should be augmented without imposing high additional fees of services offloading. Our proposition considers a model of numerous multi-persona devices held by an institution's end users. It includes a smart decision maker, centralized remotely, which in one hand analyzes the profile of each and every multi-persona device as well as the usage fees of remote resources, and on the other hand capable of minimizing the evaluation overhead which can form itself a bottleneck on the mobile terminal. Differently from all works studied in the literature, our proposition applies optimization on different extents. It includes a two-level optimization model that considers personas on all devices involved while analyzing the energy consumption, CPU usage, memory consumption and performance of each of their components in addition to the monetary cost accordingly. The proposed solution decodes, on every device engaged, the dissemination of computations that achieves a balance between multi-persona devices needs and corresponding cost, reaching not only a balance between high savings in local resource usage and execution speed-up but also significant satisfaction level of multi-persona devices involved and considerable reduction in the monetary charges. Moreover, we redesign a genetic-based algorithm with tailored adaptive fitness evaluation and evolution-based operator for intelligent and accelerated offloading decision making process. While centralized solution will be able to find such trade-off, clearly not all devices and services will be satisfied. Therefore besides analyzing the efficiency of this work, we examine the effect of applying our proposition in decentralized manner and give insights on the efficiency of both approaches from device user and institution perspectives with respect to the devices and services satisfaction level and the overall cost.

4.5 Illustrative Business Model and Problem Description

Running multiple personas on a mobile device is challenging due to the needed additional virtualization layer on such resource constrained terminal. The limited CPU power, memory, and battery can impose serious performance overhead, decrease the responsiveness and put

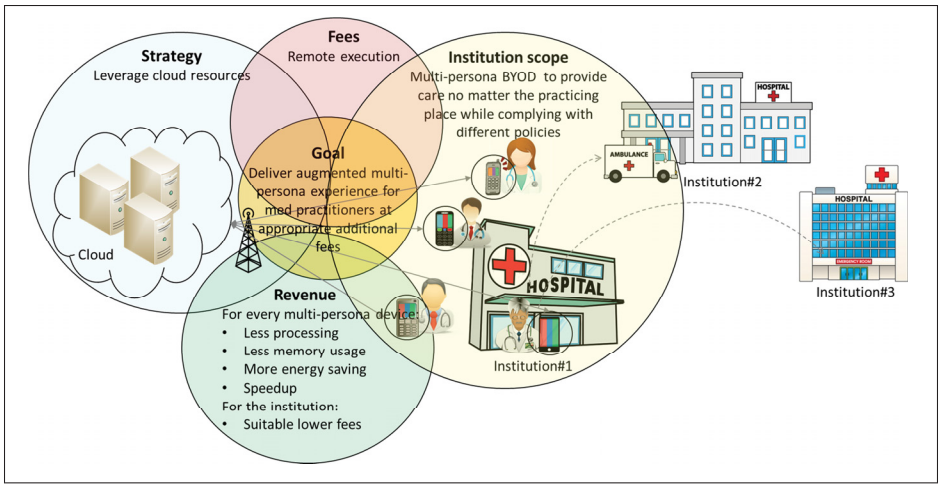


Figure 4.2 Illustrative business model.

the viability of personas on the line. We proposed in previous work an offloading-based solution to deal with these issues [Tout *et al.* (2016)]. Yet, leveraging cloud resources to support multi-persona in workplace raise new problems as different requirements should be taken into account. To emphasize these concerns, we consider the business model illustrated in Figure 4.2. It's worth to mention that the following concerns apply on other business models mainly in enterprises where multi-persona BYOD is embraced by employees to balance personal and professional facets, while the healthcare system is just used here as an illustrative example. The demands of medicine impose on doctors to move from traditional private practice to be also part of larger care provider institutions. Working at different hospitals allows doctors to be a point of continuity for more patients, providing them with coverage from diagnosis to remedy. Yet, doctors are subject to different policies that reflect each institution. Yet, with a persona for each, doctors are able to comply with each policy while providing continuous care for their patients no matter their place of practice. However, with several multi-persona devices used by medical practitioners are now involved, leveraging cloud resources to deliver augmented multi-persona experience on the terminal imposes significant additional fees, as many offloading service requests impose higher expenses. Therefore, every time an offloading decision is to be taken, it should balance the resource savings and performance acceleration on the concerned devices with the monetary fees of remote resources leveraged to carry out these benefits.

We shed the light on this problem by showing the advantages of offloading on the end terminal yet the additional monetary fees it imposes on the institution in return. With no available standards testbeds for such evaluation, the mobile applications used in these experiments implements algorithms for virus scanning and encryption and a variety of games, namely, nqueens puzzle, sudoku solver and maze discoverer. To simulate different range of offloading requests that can be generated by end users devices in an institution, we vary the number of devices with three personas on each, all running these applications. To study the impact of offloading, we compare two strategies; ALLoc where applications on all devices are running locally and ALLO where they are offloaded to be executed on a remote infrastructure. We measure the average CPU usage, memory consumption, energy loss and execution time, per device as well as the overall monetary cost imposed. Computation resources are typically provided in the form of virtual machine (VM) instances and billed based on usage time [Amazon (b); Google (b)]. The calculated cost in this work is based on the on-demand pricing scheme of a VM instance of type m4.4xlarge from Amazon [Amazon (b)], which has been selected with no particular preferences as any instance from commercial providers would fulfill the objective of these experiments. Details about the tools are described in Section 4.9.

The results are depicted in Figure 4.3. The ALLoc strategy imposed considerable overhead on every user terminal with up to 76% CPU usage, 30.6% memory consumption, 1557.7 KJ of energy loss and poor performance with 39 min to finish the execution. In contrast, ALLO was able to significantly minimize the resource usage with just 0.9% of the CPU usage, 14.8% memory, 4100 J of energy and offer enhanced performance with 15 minutes of execution time. However, the results show also the additional high charges of ALLO needed for the institution to offer such benefits to its medicine practitioners. The fees varied between 3 and 24$ per hour. The incremental value of fees in ALLO strategy is due to the offloading requests sent by each device as in this scheme, all applications are executed remotely. For ALLoc, one can notice stable results even when incrementing the number of devices, this is due to the fact that all devices are running the same applications and following the same strategy of local execution. Similar stability for ALLO strategy, yet this is interpreted by the fact that the setup scales

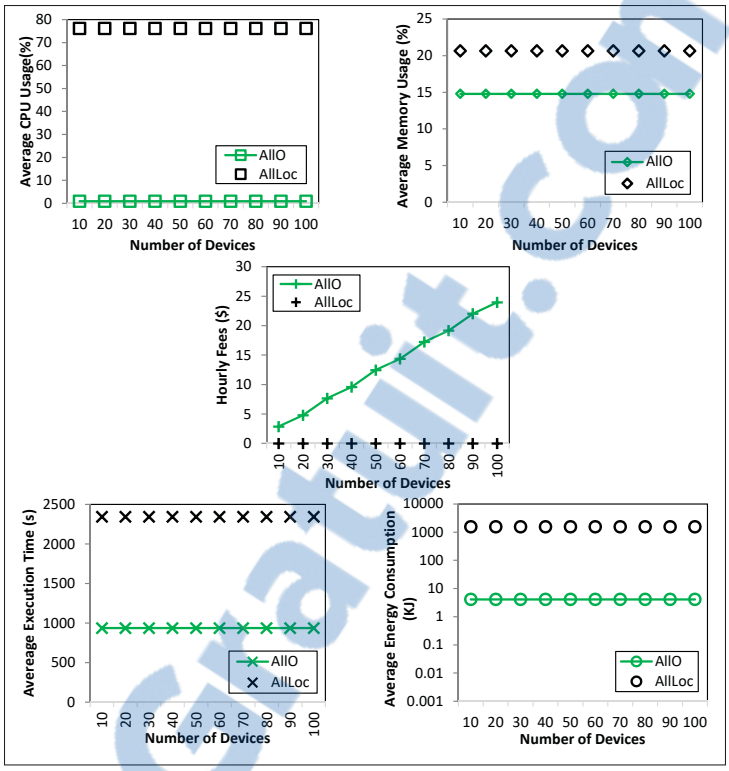


Figure 4.3 Unbalanced resource usage, performance, and monetary fees.

horizontally to avoid any influence on the server's throughput. The findings in Figure 4.3 raise the need for a new strategy that is able to balance, resource usage and performance to boost personas user experience on different devices and the remote resources usage to minimize the additional fees implied.

4.6 Cost-Effective Offloading: System Model

The system model is depicted in Figure 4.4. With multi-persona adopted in workplace, many multi-persona devices are engaged. We devote per persona profilers on each device to monitor the relevant resources. The profilers gather information about CPU and memory usages, energy consumption, and execution time for all components. A component can be an application, service, method, or even thread that implements certain functionalities. The profilers examine also the availability of connectivity, its data rate and latency in the host persona.

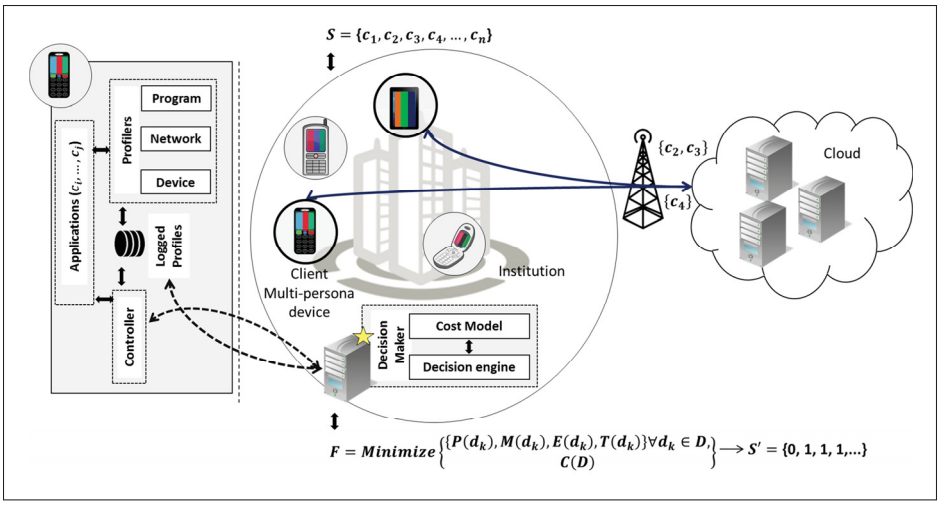


Figure 4.4 System model.

Profilers on every device communicate the gathered information with a centralized smart decision maker hosted and managed by the institution to construct and resolve two-level optimization model whose solution balances the fundamental requirements of personas on various users devices and the implied fees. The profiled information forms not only the input for the decision maker but also the trigger for offloading evaluation. With periodic monitoring, the host device will be able to send a request for the decision maker to analyze the benefits and costs of any decision before personas fall in critical states. Devices resources and performance, and remote resource fees, all constitute the actual cost. Therefore we propose a decision model composed of different metrics that in one hand affect resources and performance of personas, and on the other hand the total remote resource usage fees for executing offloaded components. In this regard, we devise a two-level multi-objective optimization model that considers CPU usage $P(d_k)$, memory consumption$M(d_k)$, energy loss $E(d_k)$ and execution time $P(T_k)$ for personas on each and every device, as well as the remote execution fees $C(D)$; $(F = Minimize\{P(d_k), M(d_k), E(d_k), T(d_k) \forall d_k \in D, C(D)\})$. Each of these metrics is weighted in order to comply with different states on the device. The formal design of the proposed model and more details are presented and interpreted in Section 4.7.

The Controller is mainly responsible of enforcing privileges between personas. Typically, professional persona has more priority to offload its computation than personal one. While when

it comes to multiple professional personas, the one that belongs to the institution where the device is being used at the time being, can be given higher priority. Yet such assumption does not always hold. For instance, doctors use their multi-persona devices to provide care for their patients no matter their place of practice, therefore the controller allow them to manage such privileges between personas, and at lower level to enforce them for particular tasks to handle critical patients situations. Other policies that are subject of future work include shutting down particular personas or applications when the device is running out of resources or suffers from performance degradation.

Devices are not independent as in the sense of resource sharing, the sum of offloading fees is one of the multiple objectives. However, this two level of objectives are inter-conflicting. With the aid of offloading to optimize the limited resources and performance of multi-persona devices, additional fees are imposed, which emphasizes the complexity of finding a trade-off among these objectives. For this end, we devote a genetic-based algorithm with tailored evolution elements to generate the distribution of components, which optimizes the global cost. The optimal distribution ($S' = \{0,1,1,1,...\}$) dictates for personas components on every device, whether to be executed locally (0 value) or offloaded (value of 1) like c_2, c_3, c_4. The core details of the decision making process are presented in Section 4.8.

4.7 Collective Multi-Persona Offloading Optimization Problem (CMPO)

4.7.1 Definition

Differently from [Tout *et al.* (2016)], when many devices need to offload components to boost personas performance and ensure longer viability, the offloading decision of each influences the other as requested remote resources are shared among them and the monetary fees of the latter cannot be disregarded. This requires finding the optimal dissemination of components with the lowest-cost for multi-persona mobile devices in one hand and the institution spending on the other hand. We represent the set of multi-persona devices as D, where each device is defined as $d_k/k = [1,...,r]$ and r is the number of devices, personas as $P = \{p_1,...,pm\}$, the components

set as $S = \{c_1, ..., c_n\}$ and the disseminations solutions set as G. Here, we consider the cost as every mobile resource usage in terms of processing $P(d_k)$, memory $M(d_k)$ and energy $E(d_k)$ besides its performance $T(d_k)$ as well as the total monetary fees of remote executions $C(D)$; The CMPO optimization problem can be defined as follows:

Assumptions. We assume in this problem that offloading can be prioritized between personas. Each of the latter has a weight that gives priority for its components to be offloaded. These weights can be assigned by multi-persona device user and enforced by the controller module. Such weights can give priorities for professional personas over personal ones or foreground over background to use cloud resources, or even allow doctors to prioritize offloading from particular persona that deals with critical patient case. Additionally, components are assumed to be independent and some of them might not be offloadable. Studying the dependability between components form part of future work where such relation has direct impact on their execution. We also consider dynamic environment where the network might not be available and its data rate and latency may vary.

Definition 1. *Given various devices $d_k \in D/k = [1, ..., r]$ of multiple personas $P = \{p_1, ..., p_m\}$ characterized by $power_{cpu}$, $power_{screen}$, $power_{idle}$, $power_{transmission}$, which correspond to powers consumed on processing, screen brightness, idle cpu and for transmission, and running set of components $S = \{c_1, ..., c_n\}$ each of which c_i uses $CPU_{c_i}^{local}$ percent of the CPU, $Memory_{c_i}^{local}$ percent of the memory, consumes $Energy_{c_i}^{local}$ energy, spends $ExecutionTime_{c_i}^{local}$ time, when it is executed locally whereas $CPU_{c_i}^{remote}$, $Memory_{c_i}^{remote}$, $Energy_{c_i}^{remote}$ and $ExecutionTime_{c_i}^{remote}$ respectively when it is offloaded and relevant data to be transmitted accordingly $Data_{c_i}$, while the network features are Bandwidth and Latency; find the dissemination of components S' that optimizes $P(d_k)$, $M(d_k)$, $E(d_k)$ and $T(d_k)$, $\forall d_k \in D$, with minimal $C(D)$.*

The optimization objectives in this problem present both inter and intra levels of conflicts. At high level, the aim to optimize the resource usage and performance on every device, by offloading the execution of components in their host personas, and minimize the monetary fees of remote execution are inconsistent intents. Likewise, at the low level of every multi-persona de-

vice, speeding up the execution while minimizing processing, memory usage and energy consumption are also paradoxical. Further, finding the optimal dissemination that balances these objectives suffers from an exponential search space in the number of different possibilities in which these components can be distributed, which increases the complexity of the problem. This is similar to the various ways n distinct objects can be distributed into m different sacks with k_1 objects in the first sack, k_2 in the second, etc. and k_1+k_2+....k_m= n. This indeed is obtained by applying the multinomial theorem where $(k_1+k_2+...k_m)^n = \sum \binom{n}{k_1 k_2 .... k_m} k_1^{n_1} k_2^{n_2} ... k_m^{n_m}$. Here, the objects are the components on different multi-persona devices and the sacks are the mobile device and the remote infrastructure. Thus, for n components, there are 2^n different dissemination possibilities.

Theorem 1. The CMPO optimization problem is NP-Hard

Proof:*Next, we will prove that the collective multi-persona offloading optimization problem is NP-Hard via a reduction from the NP-hard multi-objective-m-dimensional Knapsack Problem (MOMKP) [Lust & Teghem (2012)]. We provide the following definition of the MOMKP: Let n denote the number of items and m the number of knapsacks. Consider* p_i^j *and* w_i^j*, the profits and weights of item i with respect to the knapsack j respectively, and* c_j*, the capacity of knapsack j. Define the solution as* $x = \{x_1, ..., x_n\}$ *with* $x_i \in \{0,1\}$*. The objective is to distribute the items into the knapsacks in a way to maximize* $f(x) = (f_1(x), f_2(x), ..., f_k(x)), where f_i(x) = \sum_{j=1}^{n} p_i^j . x_i$ *subject to* $\sum_{j=1}^{m} w_i^j . x_j < c_i \forall i \in \{1, ..., m\}$.

Now given an instance of the MOMKP, we transform it into an instance of CMPO problem as follows: Let n denotes the number of components from different multi-personas devices and m the number of knapsacks, which is here equal to two (mobile device for local execution and remote infrastructure for offloaded execution). Consider $CPU_{c_i}^{local}$*,* $Memory_{c_i}^{local}$*,* $Energy_{c_i}^{local}$ *and* $ExecutionTime_{c_i}^{local}$ *as processing, memory, energy and execution time costs of component* c_i*, in persona* p_j *on device* d_k*, respectively when it is executed locally (i.e.,* c_i *cost with respect to* $sack_1$*), while* $CPU_{c_i}^{remote}$*,* $Memory_{c_i}^{remote}$*,* $Energy_{c_i}^{remote}$ *and* $ExecutionTime_{c_i}^{remote}$ *when it is offloaded (i.e.,* c_i *cost with respect to* $sack_2$*). In the latter case, a monetary cost* c_{c_i} *for remote*

execution is also imposed. For every device d_k, let $P(d_k)$, $M(d_k)$, $E(d_k)$ and $T(d_k)$ be the total cost on every device, which are the sum of the costs of its underlying local and remote components form every persona. The total monetary cost is the aggregation of costs of offloaded components from each device, which is denoted as C_D, where D is the set of devices. Each sack has its own capacity, thus every d_k has $\tilde{T}_{P_{d_k}}$, $\tilde{T}_{M_{d_k}}$, $\tilde{T}_{E_{d_k}}$ to guarantee enough resources for personas components. Here we assume that the remote sack has enough resources to support the mobile devices requests as long as the institution is capable of devising the adequate needed resources. Define the solution as $x = \{x_{c_1}, ..., x_{c_n}\}$ with $x_{c_i} \in \{0,1\}$, where a bit of 0 denotes local execution and a bit of 1 is for remote processing. The question is how to disseminate components in each persona on every device between local and remote infrastructure in a way minimize$(P(d_k)\forall d_k \in D, M(d_k)\forall d_k \in D, E(d_k)\forall d_k \in D, T(d_k)\forall d_k \in D, C(D))$ subject to $P(d_k) < \tilde{T}_{P_{d_k}} \forall d_k \in D$, $M(d_k) < \tilde{T}_{M_{d_k}} \forall d_k \in D$, $E(d_k) < \tilde{T}_{E_{d_k}} \forall d_k \in D$, and $C(D) < B$. Accordingly, a solution of CMPO yields a solution to the MOMKP. After this complete proof of the reduction, we conclude that the CMPO problem is NP-Hard.

4.7.2 Formulation

In the following, we mathematically formulate CMPO based on the definitions we provided.

- Decision Variables:

$$x = \{x_{c_1}, ...x_{c_n}\}$$

where,

$$\forall c_{i,i:1\to n}, x_{c_i} = \begin{cases} 0, & \text{if } c_i \text{ is to be executed locally} \\ 1, & \text{if } c_i \text{ is to offloaded} \end{cases}$$

- Parameters:

- Mathematical Model:

$Minimize((P(d_k), M(d_k), E(d_k), T(d_k))\forall d_k \in D, C(D))$ where,

$$P(d_k) = w_{P(d_k)} \times \left[\sum_{\forall p_j \in d_k} w_{p_j} \sum_{\forall c_i \in p_j} (1 - x_{c_i}) \times CPU_{c_i}^{local} + \sum_{\forall p_j \in d_k} \sum_{\forall c_i \in p_j} \gamma_{c_i} x_{c_i} \times (CPU_{c_i}^{remote}) \right] \tag{4.1}$$

D	set of devices
d_k	device in D
p_j	persona on device d_k
c_i	component in persona p_j
γ_{c_i}	offloadable component indicator
$CPU_{c_i}^{local}$	cpu usage on d_k by c_i executed locally
$CPU_{c_i}^{remote}$	cpu usage on d_k by c_i offloaded
$Memory_{c_i}^{local}$	memory usage on d_k by c_i executed locally
$Memory_{c_i}^{remote}$	memory usage on d_k by c_i offloaded
$power_{cpu}$	power consumed by d_k on precessing
$power_{screen}$	power consumed by d_k on the screen
$power_{idle}$	power consumed by d_k on idle CPU
$power_{transmission}$	power consumed by d_k for transmition
$Data_{c_i}$	size of data transmitted for offloading c_i
$ExecutionTime_{c_i}^{local}$	time to execute c_i locally
$ExecutionTime_{c_i}^{remote}$	round trip time to process c_i remotely
$price$	hourly monetary cost of remote execution
w_{p_j}	weight for persona p_j
$w_{P(d_k)}$	weight for function $P(d_k)$
$w_{M(d_k)}$	weight for function $M(d_k)$
$w_{E(d_k)}$	weight for function $E(d_k)$
$w_{T(d_k)}$	weight for function $T(d_k)$
$w_{C(D)}$	weight for function $C(D)$

$$M(d_k) = w_{M(d_k)} \times \left[\sum_{\forall p_j \in d_k} w_{p_j} \sum_{\forall c_i \in p_j} (1 - x_{c_i}) \times Memory_{c_i}^{local} + \sum_{\forall p_j \in d_k} \sum_{\forall c_i \in p_j} \gamma_{c_i} x_{c_i} \times (Memory_{c_i}^{remote}) \right] \tag{4.2}$$

$$E(d_k) = w_{E(d_k)} \times \left[\sum_{\forall p_j \in d_k} w_{p_j} \sum_{\forall c_i \in p_j} (1 - x_{c_i}) \times \Big((power_{cpu} + power_{screen}) \times ExecutionTime_{c_i}^{local} \Big) + \right.$$

$$\left. \sum_{\forall p_j \in d_k} \sum_{\forall c_i \in p_j} \gamma_{c_i} x_{c_i} \times \left((power_{idle} \times ExecutionTime_{c_i}^{remote}) + \left(power_{transmission} \times (Latency + \frac{Data_{c_i}}{Bandwidth}) \right) \right) \right] \tag{4.3}$$

$$T(d_k) = w_{T(d_k)} \times \left[\sum_{\forall p_j \in d_k} w_{p_j} \sum_{\forall c_i \in p_j} (1 - x_{c_i}) \times ExecutionTime_{c_i}^{local} + \sum_{\forall p_j \in d_k} \sum_{\forall c_i \in p_j} \gamma_{c_i} x_{c_i} \times (ExecutionTime_{c_i}^{remote} \right.$$

$$\left. + Latency + \frac{Data_{c_i}}{Bandwidth}) \right] \tag{4.4}$$

$$C(D) = w_{C(D)} \times \left[\sum_{\forall d_k \in D} \sum_{\forall p_j \in d_k} w_{p_j} \sum_{\forall c_i \in p_j} x_{c_i} \times (ExecutionTime_{c_i}^{remote} \times price) \right] \tag{4.5}$$

Subject to

$$P(d_k) < \tilde{T}_{P_{d_k}} \forall d_k \in D \quad (c_1)$$
$$M(d_k) < \tilde{T}_{M_{d_k}} \forall d_k \in D \quad (c_2)$$
$$E(d_k) < \tilde{T}_{E_{d_k}} \forall d_k \in D \quad (c_3)$$

These equations stimulates the model to find the dissemination of components that can balance the resource usage and performance for as many services as possible on the multi-persona mobile devices and the additional monetary fees imposed for offloaded components. This is a two-level optimization model that considers minimizing the resource consumption and boosting the performance on as many involved multi-person mobile devices as possible, through computation offloading, while takes also into account a conflicting metric of reducing the monetary fees entailed by using the remote resources for offloading requests. We formulate CMPO as multi-objective optimization through the **minimization function** F which includes five objectives aiming to reduce the resource consumption, execution time and fees implied.

$\forall d_k \in D$, $P(d_k)$ in Equation (4.1) is calculated as the processing needed to execute local services and that required during the execution of remote computations and/or for processing the returned response. As $\forall d_k \in D$, $M(d_k)$ in Equation (4.2) is determined by the memory needed to process local computations and the one consumed while waiting and/or processing the remote components execution response. $\forall d_k \in D$, $E(d_k)$ in Equation (4.3) is the energy consumed on CPU processing and screen brightness for local components execution and the energy spent on the CPU being idle, screen brightness and network being active while waiting the offloaded services execution. It also includes the energy consumed by the terminal for data transmission of relevant uploaded parameters and downloaded response. $\forall d_k \in D$, The time needed to finish the execution of requested services on every device is defined by $T(d_k)$ in Equation (4.4), which includes the duration of local and remote processing in addition to any latency and data transmission involved. As for the second layer of this model, Equation (4.5) defines the monetary fees imposed for processing offloading requests from relevant multi-persona devices, which is calculated based on the execution time needed for remote processing. Constraint (c_1) forces the model to find a solution that does not overload the processing on the mobile terminal. Constraint (c_2) forces the model to find a solution that does not excess the memory of the

local device. Constraint (c_3) represents the energy constraint that guarantees a threshold for available power on the device.

4.8 Smart Cost-Effective Decision Maker

As our CMPO is NP-Hard, we propose a heuristic based algorithm to solve it. The proposed smart decision maker (SDM) exploits the intelligent evolution of solutions in genetic algorithms (GAs), which have been able to solve complex optimization problems in many areas through their method of evolution inspired search [Deb (1999)]. Based on natural selection, GAs simulate the propagation of the fittest individuals over consecutive generations to determine the best solution. After investigating different algorithms, an extensive study presented in previous work [Tout *et al.* (2016)] proves the efficiency of nsga-ii [Deb *et al.* (2002)] over the others. Differently from [Tout *et al.* (2016)], we redesign nsga-ii [Deb *et al.* (2002)] with tailored adaptive fitness evaluation and evolution-based crossover. Particularly, according to the proposed two-level optimization model that we presented in Section 4.7, SDM works on generating the dissemination of personas components that minimizes the resource usage and enhances the performance on every device with minimal additional fees for the institution. In addition, we propose an evolution-based crossover operator aiming to accelerate the detection of favorable candidate solutions that optimize CMPO. Based on the differential evolvement of the individuals, genes that produce better fitness compared to their parents are used to form the offspring. Algorithm 4.1 describes the process adopted by the decision maker to generate cost-effective solution.

4.8.1 Solution Encoding

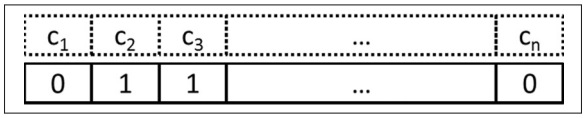


Figure 4.5 Encoding scheme.

In GAs, each chromosome/individual in a population forms a candidate solution. For CMPO, The algorithm starts with population of N randomly generated individuals according to the number of components in personas on each device (Line 5). Each individual represents the dissemination of personas components from different devices, has a size $|S|$, and it is encoded as a set of binaries $x = \{x_{c_1}, ..., x_{c_n}\}$, where n is the total number of components involved. Figure 4.5 illustrates the encoding scheme. Each gene x_{c_i} of an individual represents a persona component on a mobile device and its value dictates whether this component should be offloaded ($x_{c_i} = 1$) or executed locally on the end terminal ($x_{c_i} = 0$).

4.8.2 Fitness Evaluation

A score/fitness is designated for each individual simulating the propagation of the fittest individuals over consecutive generations in order to determine the best solution. The fitness varies based on how efficiently each individual can solve the problem. For CMPO, the fitness of each candidate solution is determined by $P(d_k)$, $M(d_k)$, $E(d_k)$, $T(d_k)$ and $C(D)$ functions (Line 6 to Line 12) that form the model proposed in Section 4.7. The solutions are ranked according to their ability to reduce the resources usage, speed-up the execution and minimize the relevant monetary fees, where all these metrics are to be minimized.

The fittest b individuals in the population are then selected (Line 15) to go through a process of evolution. In the latter, crossover and mutation operations are applied to produces next generation of individuals for new possible distribution solutions of components (Line 16 to Line 20). The algorithm reassesses the model metrics to calculate the fitness of these generated individuals (Line 21 to Line 27). The evaluation process continues over and over until any of the stopping criteria is met, where either the fitness of the best individual in successive populations did not improve or the defined number of iterations δ is reached. Finally, the decision maker returns the fittest distribution from the last generation to solve the CMPO problem (Line 35).

4.8.3 Evolution Process

4.8.3.1 Selection

In this step (Line 15), chromosomes are selected to go through the evolution process. We use bit tournament selection, which involves running several rounds over randomly chosen chromosomes from the population. The winner in each round, which has the best fitness is then selected for crossover.

4.8.3.2 Crossover

Chromosomal crossover is the operation of exchanging genes between two individuals with the intent to produce better offspring. With a μ_c rate, Crossover usually occurs when regions of a chromosome break and reconnect to the other chromosome. In contrast, we propose an evolution-based crossover operator (Line 16 to Line 18) for CMPO as illustrated in Figure 4.6. This operator is based on the differential evolution of individuals that optimizes CMPO. Taking two individuals (parents) I_1 and I_2 as depicted in Figure 4.6, highlighted genes that produce better fitness (smaller fitness value based on the evaluation presented in Section 4.8.2) when compared to their parents are used to form the offspring.

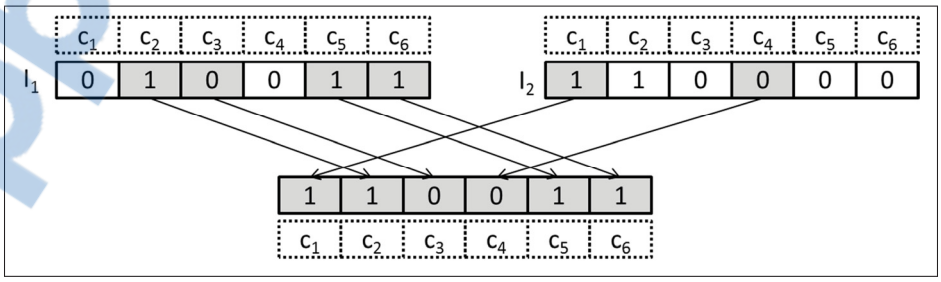


Figure 4.6 Evolution-based crossover.

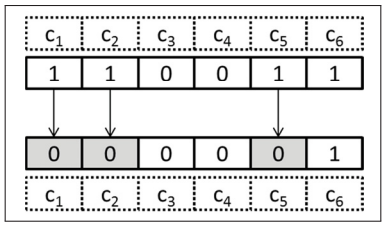


Figure 4.7 Mutation.

4.8.3.3 Mutation

Mutation operation (Lines 19 and 20) is to apply additional modifications in the chromosomes that improve their fitness. With a μ_m rate, we apply standard bit flip mutation for CMPO as depicted in Figure 4.7.

Lemma 3. The time complexity of Algorithm 4.1 is $O(\delta N|S|)$

Proof: *Different factors determine the time complexity of this algorithm; the fitness function evaluation, the population size, the individual length, variation and selection operators and the number of iterations or generations. Initializing generation index, solution set and generating the first population has each time complexity* $\mathcal{O}(1)$. *The evaluation of the fitness function has time complexity of* $\mathcal{O}(N+1)$ *where N is the population size. The tournament selection, evolution-based crossover and bit flip mutation, have time complexity of* $\mathcal{O}(\delta N|S|)$ *where* δ *is the number of generations and* $|S|$ *is the size of an individual. Reassessing the model is of* $\mathcal{O}(\delta N)$. *Finally, the return statement has* $\mathcal{O}(1)$. *Subsequently, the time complexity of the algorithm is* $\mathcal{O}(1)+\mathcal{O}(N+1)+\mathcal{O}(\delta N|S|)+\mathcal{O}(\delta N)+\mathcal{O}(1)$ *which is equivalent to* $\mathcal{O}(\delta N|S|)$ *as lower orders are to be dropped.*

4.9 Numerical Analysis

Previous examination showed that the device is not able to handle more than 2 personas with several applications running in each but rather cause the end terminal to shut down, while when adopting offloading, it was able to run up to three personas with even computation intensive apps. For more detailed results regarding how many personas and apps the device can support

Algorithm 4.1 Smart DM

Input: Set of multi-persona devices $D = \{d_1, d_2, ...d_r\}$ with DS specs of each in terms of power consumption on cpu, screen in idle state and for transmission, set of personas on these devices $P = \{p_1, p_2, ...p_m\}$, candidate components from each forming a set $S = \{c_1, c_2, ...c_n\}$, each of which is characterized by local and remote execution time, cpu usage and memory consumption, and length, network characteristics NC in terms of bandwidth and latency, instance leasing cost IC, number of possible solutions N, mutation rate μ_m, crossover rate μ_c and number of generations δ.
Output: Distribution set of components $S' \subseteq S$.
$i \longleftarrow 0$ ▷ population index
$S' \longleftarrow \emptyset$
$G_i \longleftarrow Random[N][|S|]$ ▷ random population
for $k = 1$ to r **do**
    Calculate $P(d_k)$:=$CalcProcessing(P, G_i)$
    Calculate $M(d_k)$:=$CalcMemory(P, G_i)$
    Calculate $E(d_k)$:=$CalcEnergy(DS, P, G_i, NC)$
    Calculate $T(d_k)$:=$CalcTime(P, G_i, NC)$
end for
Calculate $C(D)$:=$CalcCost(D, P, G_i, IC)$
for $g = 1$ to δ **do**
    {
    Propagate b best candidates distributions BC for next generation $G_{i+1} \longleftarrow BC$
    select two solutions from G_i, X_A and X_B;
    Generate X_C by evolution-based crossover to X_A, X_B;
    Add X_C to G_{i+2};
    Select a solution X_b from G_{i+2};
    Mutate X_b and generate new feasible solution $X_{j'}$;
    for $k = 1$ to r **do**
        Reassess $P(d_k)$:=$CalcProcessing(P, G_{i+1})$
        Reassess $M(d_k)$:=$CalcMemory(P, G_{i+1})$
        Reassess $E(d_k)$:=$CalcEnergy(DS, P, G_{i+1}, NC)$
        Reassess $T(d_k)$:=$CalcTime(P, G_{i+1}, NC)$
    end for
    Reassess $C(D)$:=$CalcCost(D, P, G_{i+1}, IC)$
    Update generation $G_i = G_{i+1} + G_{i+2}$
    Update generation index $i \longleftarrow i + 1$
    if Same fitness is detected in G_{i+1} and G_{i+2} **then**
        break;
    end if
    }
end for
return most cost-effective distribution S' from G_i

with and without offloading, one can refer to [Tout *et al.* (2016)]. We interpret in this Section the experiments performed to answer the following fundamental research questions: (1) How efficiently the proposed two-level multi-objective optimization model and the cost-effective smart decision maker are able to generate a trade-off between the usage of cloud resources for

minimal processing, memory, energy and execution time in personas on different mobile terminals, and the additional remote execution fees imposed accordingly? (2) What is the satisfaction rate achieved by centralized and decentralized applications of our proposition respectively and which of them would be more competent from both users and institution perspectives? (3) What kind of improvements are achieved through the optimized decision engine?

4.9.1 Setup

The implementation of a mobile application should follow particular design pattern in order to make it offloadable. The activity/service model in android allows clear separation between the application code and its user interface. Particularly, the logic code of the computation-intensive tasks can be implemented as services through android interface definition language (AIDL) while the user interface is defined using activities. Applying such model facilitates the offloading task as services and activities are already isolated. We use the same applications described in Section 4.5 to run on the mobile devices. Each mobile device is a three-persona terminal running Android operating system with quad-core processor and 1 GB of RAM.

To profile the running applications, we implemented Linux-based commands that monitor CPU and memory usages, while we used PowerTutor [Zhang *et al.* (2010)] to monitor the energy consumption on different aspects like CPU, screen and network. As for the power consumption on idle CPU, active network and during transmissions, we exploit the power profile in android [Android (2017)] to get the relevant information. Finally, we embedded a timer to monitor the execution time. On first invocation, the offloadable services are sent for remote execution on pre-configured server. Only class files of the remote implementation are automatically packaged as jars and transmitted to the server. Whenever the requested services are already hosted on the server, only relevant parameters are communicated. Yet, the relevant overhead is included in the results. The data gathered by the profilers is logged and fed as input to the decision maker. The connection between local and remote resource is achieved through WiFi network in infrastructure mode, characterized by the IEEE 802.11n wireless networking standard. In these experiments, WiFi is used as it is typically adopted in an institution to

connect mobile devices to the internet, however other networks can be used in real-life setup like 3G and LTE. As for the decision algorithm, the configuration is based on empirical study of 50 times of execution. Accordingly, we set $\mu_c = 0.6$, $\mu_c = 1/n$, where n is the number of decision variables, $N = 100$ and the number of generations δ to be 20, 45, 70, 75, 100, 135, 135, 170, 185, 200 for 10 up to 100 devices respectively. The calculated monetary cost here is based on the pricing of on demand m4.4xlarge VM instance from Amazon [Amazon (b)] running Ubuntu 12.04 with 7.3 GB memory and quad core AMD Phenom(tm) II X4 B95 processor.

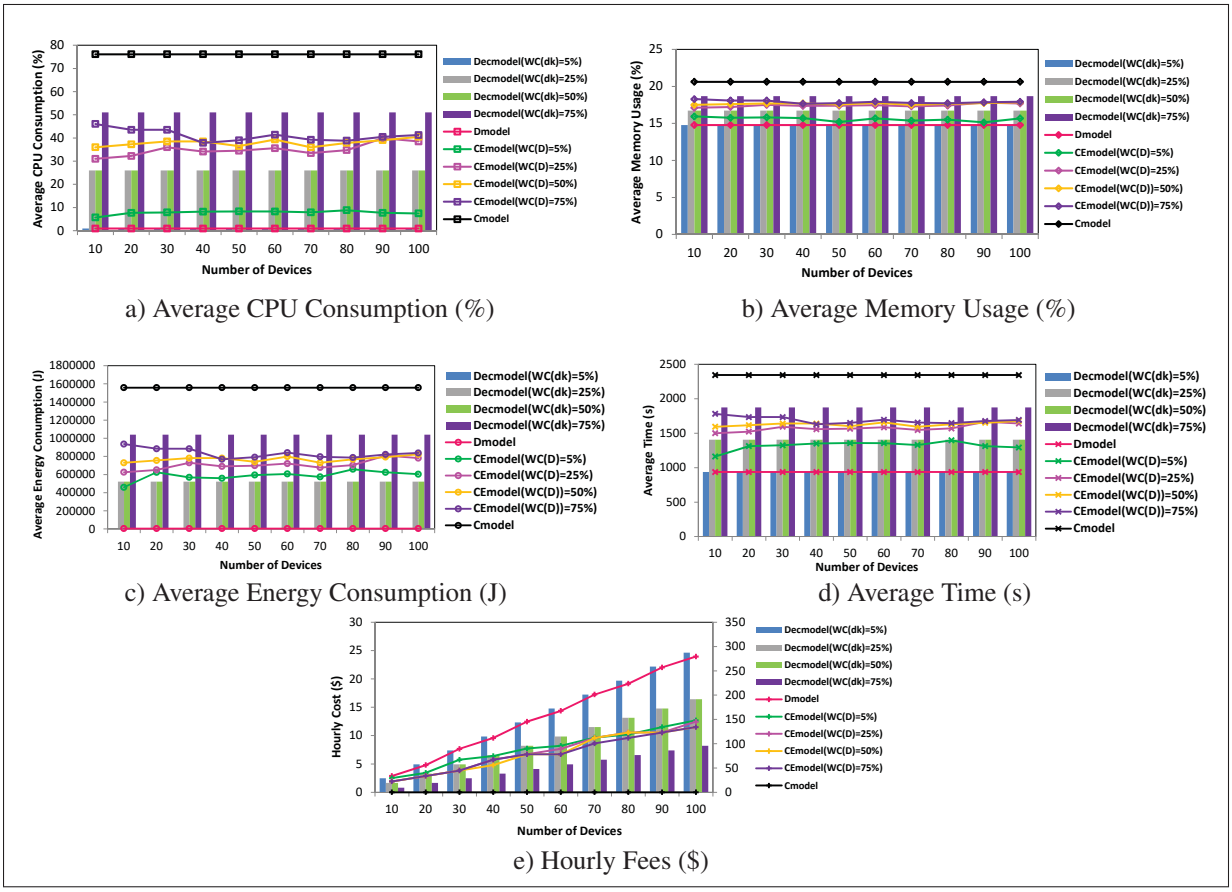


Figure 4.8 Generated solution cost: average local CPU usage, memory consumption, execution time, energy loss and monetary fees.

4.9.2 Generated distribution cost

In the first set of experiments, we compare the proposed cost-effective solution applied in both centralized and decentralized manners, CEmodel and Decmodel respectively, along with two other base approaches namely, Dmodel and Cmodel. We consider four different weights assigned to the monetary fees metric $w_{C(D)} = 75\%$, $w_{C(D)} = 50\%$, $w_{C(D)} = 25\%$ and $w_{C(D)} = 5\%$, while distributing the remaining weights equally between the other metrics, i.e., $(1 - w_{c(D)})/4$. Such distribution allows to study the effect of the assigned weights on the generated solution cost.

- CEmodel: proposed centralized solution.
- Decmodel: proposed solution with cost model evaluated independently on each device with no knowledge about other devices personas.
- Dmodel: deploying model that does not consider the monetary fees in its decision making process.
- Cmodel: deploying model that neglects the multi-persona devices needs in the evaluation process.

The results are depicted in Figure 4.8. The decision maker in Dmodel dictated all services from all personas on every device to be offloaded, typically as this model does not consider the tariff of remote execution. The numerical results show that such decision is able to mostly minimize the average processing to 0.9% (Figure 4.8a), the average memory usage to 15% (Figure 4.8b), the average execution time to 15.6 minutes (Figure 4.8d) and the energy loss to 4103J (Figure 4.8c). Yet to achieve such gain, this decision imposes between 3 and 24$ per hour (Figure 4.8e) according to the number of devices. Per contra, with Cmodel considering only the remote fees while disregarding personas metrics on the device, dictates all services to run locally, which though generates zero fees (Figure 4.8e), it shows high resources exhaustion on the terminals with up to 76% CPU usage (Figure 4.8a), 20% of memory consumption (Figure 4.8b), 39 minutes to finish the execution (Figure 4.8d) and 15.5KJ of energy loss (Figure 4.8c). However,

when it comes to the proposed model, our findings illustrates its ability to offer good balance between personas metrics on the host terminals and the monetary fees for remote resources, while varying the $w_{C(D)}$ between 25%, 50% and 75%, yet the best trade-off is shown with $w_{C(D)} = 5\%$. In the latter case, the results show an averages of CPU usage between 5%-8% (Figure 4.8a), memory usage between 15%-16% (Figure 4.8b), execution time 19-22 minutes (Figure 4.8d) and energy consumption of 459KJ-657KJ (Figure 4.8c). These improvements are close to those of Dmodel with acceptable overhead compared to the significant tariff minimization that our proposition is able to offer, varying between 2$ and 12$ with respect to the number of devices (Figure 4.8e).

We also studied the efficiency of decentralized evaluation of our model. Decmodel with 5% cost weight gives the best savings in terms of mobile device resources with 0.9% of CPU usage (Figure 4.8a), 14% of memory consumption (Figure 4.8b), 4KJ of energy (Figure 4.8c) and 15 min for applications execution (Figure 4.8d). However it is also noticed that Decmodel imposes high cost which varies between 28 and 287$ per hour, while CEmodel is able to achieve close savings results with way less fees varying between 2 and 12$ according to the number of devices involved (Figure 4.8e).

4.9.3 Decision maker overhead

This experiment aims to study the overhead of centralized and decentralized offloading decision making. In a centralized approach, the decision maker is running on a remote server yet, in a centralized one, the processing is done on each mobile terminal independently. The results are depicted in Figure 4.9. Though processing higher number of offloading service requests necessitates additional time, the augmentation was marginal compared to the additional number of devices involved. With just an increase from 0.01 to 0.2 seconds along with 10 times additional devices (Figure 4.9d), centralized remote DM releases the mobile terminal from intensive processing negligible additional overhead for the evaluation process in terms of CPU (Figure 4.9a), memory (Figure 4.9b) and energy consumption (Figure 4.9c)on the mobile terminal. These results can be interpreted by the fact that running the DM remotely on central-

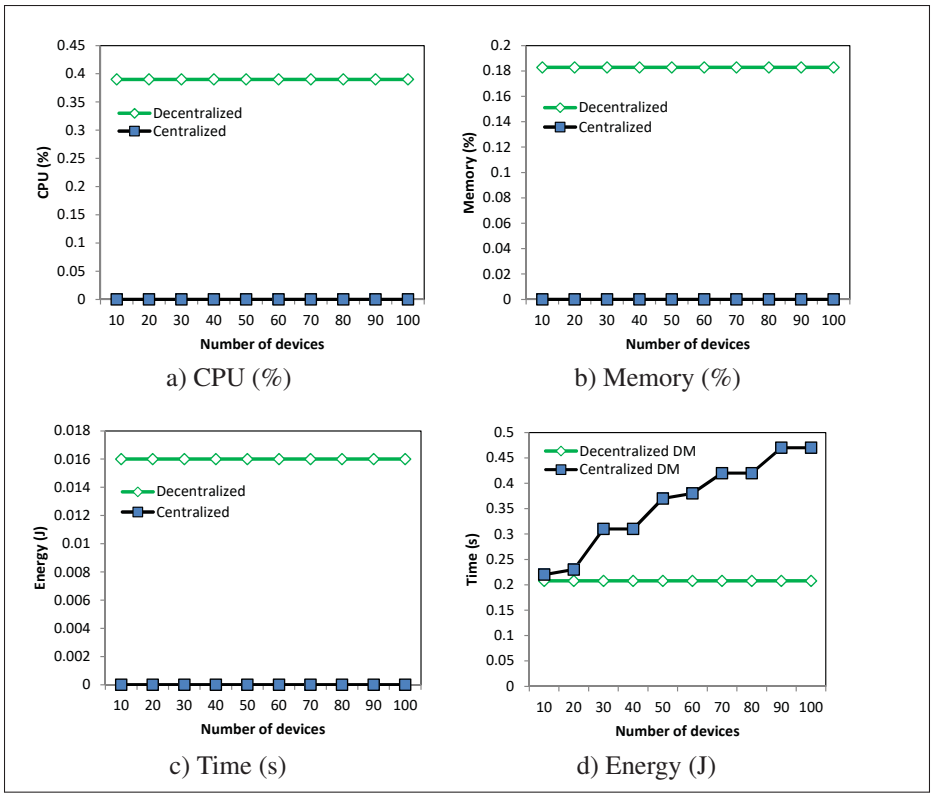


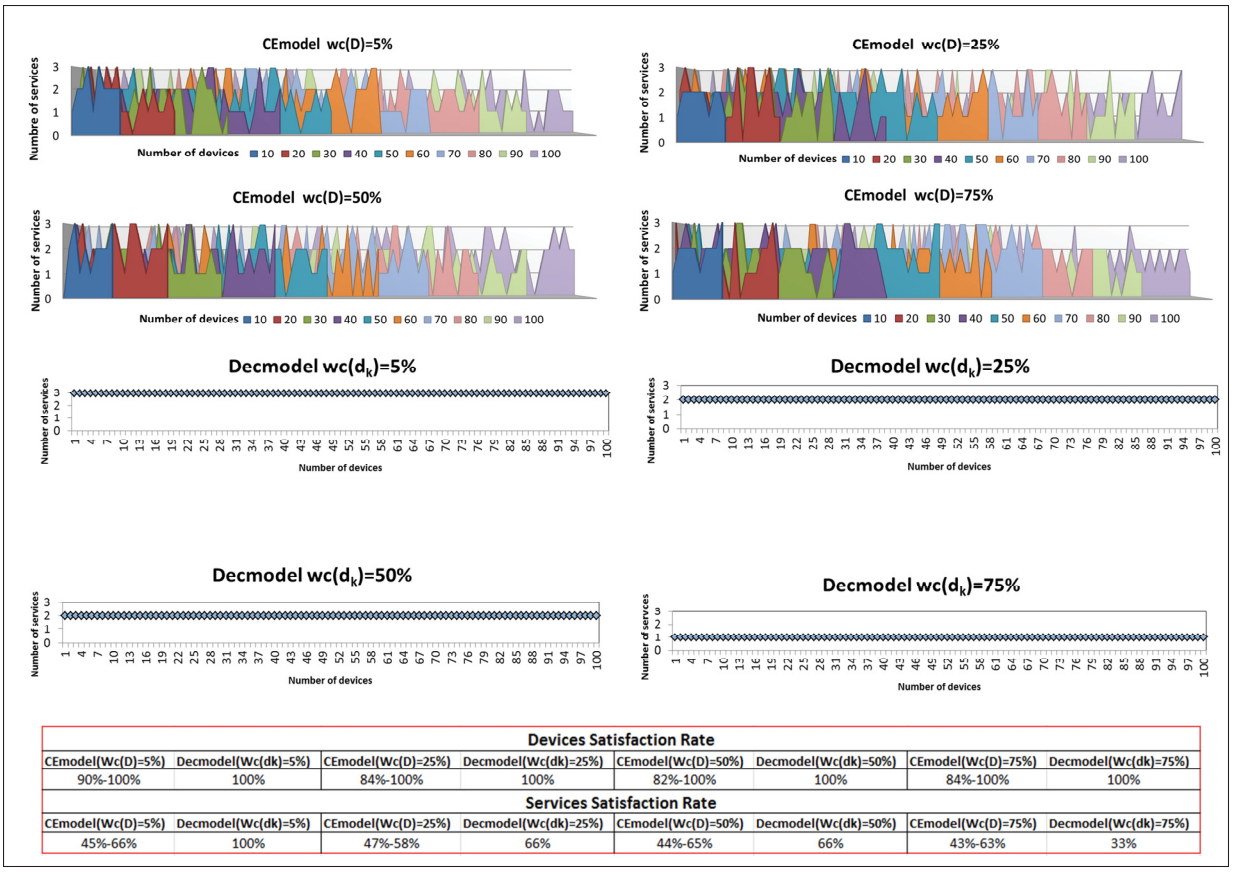
Figure 4.9 Decision maker overhead.

ized entity releases the mobile devices resources with additional yet marginal time needed to generate the decision, compared to the decentralized execution where each mobile terminal is responsible of running the offloading evaluation process locally and independently.

4.9.4 Satisfaction rate

These experiments aim to compare the satisfaction rate that can be achieved through centralized and independent offloading service evaluation. We study the satisfaction at both mobile devices and services levels. A device is satisfied if at least one of its services is satisfied (in this case offloaded). Figure 4.10 illustrates the results. For the reviewers convenience, we emphasize the main findings through the highlighted table in the figure.

The results show stability in the device satisfaction rate for the independent evaluation with 100% satisfaction of Decmodel independently of the weights ranging between 5%, 25%, 50%



Devices Satisfaction Rate							
CEmodel(Wc(D)=5%)	Decmodel(Wc(dk)=5%)	CEmodel(Wc(D)=25%)	Decmodel(Wc(dk)=25%)	CEmodel(Wc(D)=50%)	Decmodel(Wc(dk)=50%)	CEmodel(Wc(D)=75%)	Decmodel(Wc(dk)=75%)
90%-100%	100%	84%-100%	100%	82%-100%	100%	84%-100%	100%
Services Satisfaction Rate							
CEmodel(Wc(D)=5%)	Decmodel(Wc(dk)=5%)	CEmodel(Wc(D)=25%)	Decmodel(Wc(dk)=25%)	CEmodel(Wc(D)=50%)	Decmodel(Wc(dk)=50%)	CEmodel(Wc(D)=75%)	Decmodel(Wc(dk)=75%)
45%-66%	100%	47%-58%	66%	44%-65%	66%	43%-63%	33%

Figure 4.10 Satisfaction rate.

and 75% respectively, yet not to forget the high additional fees it imposes in return as shown in Figure 4.8e. As for the centralized approach, CEmodel, the results show its ability to reach high satisfaction rate yet with instability in the results with 90%-100% satisfaction for $w_{C(D)} = 5\%$, 84%-100% for $w_{C(D)} = 25\%$, 82%-100% for $w_{C(D)} = 50\%$ and 84%-100% for $w_{C(D)} = 75\%$. This can be interpreted by the aim of the centralized evaluation not only to satisfy devices collectively but also the organizations objective of minimal global additional fees. On the other hand, according to the services satisfaction rates, the results show the ability of the centralized evaluation to satisfy larger number of services on each device compared to independent evaluation when the cost metric weight increases. Particularly, with 75% weight, CEmodel reaches up to 63% services satisfaction rate while Decmodel shows only 33% satisfaction.

4.9.5 Optimized decision maker speedup

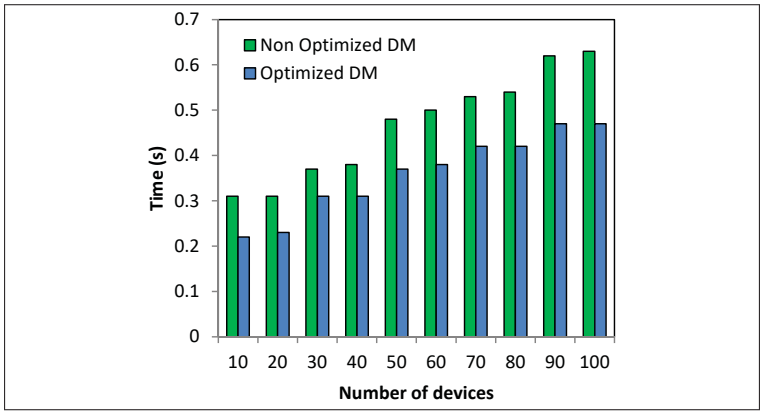


Figure 4.11 Optimized decision maker speedup.

This experiment evaluates the efficiency of the optimization introduced in the decision maker algorithm compared to the process introduced in previous achievement [Tout *et al.* (2016)]. For this end, we compare the execution time of the decision maker with and without applying the evolution based optimization. Figure 4.11 illustrates the results and proves its ability to speed-up the offloading evaluation process by 1.8 times. Such improvement is justified by the ability of the proposed differential-based operator to accelerate the convergence towards the solution that optimizes CMPO.

4.9.6 Summary

The following lessons are learned by analyzing the obtained results:

- L_1: The proposed solution is able to settle multi-persona needs on a wide range of users' terminals and the additional fees imposed on the institution on return for leveraging cloud-based offloading services. Particularly, compared to other models, our proposition proved its ability to offer a compromise that minimize processing, memory usage, energy consumption and speed-up the execution for a broad range of services on each multi-persona device with convenient additional monetary charges.

- L_2: From the institution perspective, a centralized approach is preferable as it is able to achieve high satisfaction rates along with significant reduction in the imposed fees. As from an end user perspective, a decentralized offloading evaluation is more desirable yet, just in case the number of services running on the end terminal is small not to fail with significant overhead or system survivability issue detected in previous work [Tout *et al.* (2016)].

- L_3: With more components running on the mobile device, the centralized approach proved its competency with its ability to release the terminal from considerable overhead.

- L_4: Compared to the decision maker introduced in previous contribution [Tout *et al.* (2016)], the proposed optimized decision algorithm proved its ability to speed-up the evaluation process, a critical aspect in real time mobile environments.

4.10 Conclusion and Future Directions

With the limited resources of mobile devices, offloading has proved its ability to support multi-persona. Yet, when adopted in workplace to embrace BYOD model, new problems arise. Personas on different mobile devices are now involved and share remote resources to proceed offloading service requests, which are not free of charge. The target of this paper is to find the dissemination of computations that satisfies personas needs on wider range of devices while assuring minimal additional fees for the institution imposed by remote execution. We proposed two-level multi-objective optimization model for offloading evaluation. We also proposed a smart and cost-effective offloading algorithm based on genetics with tailored evolutionary optimization. An optimal dissemination of personas components on every device offering a trade-off between end terminals and tariffs of remote processing is generated accordingly. The experimental results showed significant gain in decreasing the averages of CPU usage, memory consumption, energy loss and speeding up the execution on multi-persona devices with considerably lower monetary charges.

These results open the door for future track that examines resource management strategies to enforce policies between personas in order to reach higher satisfaction. These policies aim to manage local and remote resources by prioritizing offloading requests among personas services. This necessitates defining criticality levels and advocating prioritization scheme to classify services for offloading.

Acknowledgment

The work has been supported by École de Technologie Supérieure (ETS), NSERC Canada, the Associated Research Unit of the National Council for Scientific Research CNRS Lebanon and the Lebanese American University (LAU).

CHAPTER 5

ARTICLE 4: PROACTIVE SOLUTION AND ADVANCED MANAGEABILITY OF MULTI-PERSONA MOBILE COMPUTING

Hanine Tout[1], Nadjia Kara[1], Chamseddine Talhi[1], Azzam Mourad[2]

[1] Departement of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

[2] Department of Computer Science and Mathematics, Lebanese American University,
1102 2801 Chouran Beirut, Lebanon

Under Review

5.1 Abstract

Allowing Bring Your Own devices (BYOD) scheme in enterprises requires enforcing some policies that govern how devices will be used and how they will be managed, while maintaining end user flexibility. In order to efficiently support BYOD, mobile virtualization has become rapidly a very attractive choice as it provides both employee and enterprise with flexibility, while addressing the privacy concerns of the user and meeting the organizations security requirements. Latest lightweight mobile virtualization techniques have opened the door for multi-persona mobile computing allowing the physical device to co-host multiple virtual environments (phones). However, at any moment, the physical device resources should be enough to support the virtual instances and their applications needs to avoid performance degradation, and more critically, system crash. Many computation offloading approaches have been proposed to augment mobile devices resources. however, just an offloading strategy might not be enough, especially when some applications are not offloadable due to their criticality level or their urge to call device-related functionalities. Additionally, the overhead caused by idle applications or even idle virtual environments raise the need for more advanced strategies to complement an offloading solution. Therefore, we propose in this work a proactive solution with advanced manageability to address these issues. Through machine learning techniques,

our proposition is able to predict future context and resource needs of currently running virtual environments and potential future active ones. We also provide advanced manageability strategies that offer the ability to turn off applications and switch off virtual environments to release the physical device resources when needed. These strategies are generated based on optimization model that aims to reduce the resource usage on the physical terminal and augment its performance. A dynamic programming algorithm is proposed in order to solve this model and find the adequate management strategies. Extensive experiments were conducted and the results prove the efficiency of our proposition.

Index Terms: Mobile device, Multi-persona mobile computing, Mobile Cloud Computing, Offloading, Optimization, Dynamic Programming.

5.2 Introduction

The emergence of smartphones as the linchpin of everyday computing and communication has forced one of the most major shifts that corporates have ever seen. Mobile computing has gone from a niche market to the fastest growing, and often most popular, way to do business computing. Mobile computing is becoming not only a new computing platform, but the dominant one for many enterprises. To cope with this wave, Bring Your Own Device (BYOD) has been embraced, as a practice that allows employees to use their personal mobile devices to access corporate data and applications. The trade-off, of course, is that the corporate data therein, is being accessed and stored on personal devices, which raises many security concerns. With employee-owned devices at work, the chances of confidential company data mixing with personal employee information are high, and the instances of data leakage and loss are even higher. Along the way, corporate culture has had to change in order to accommodate the always-present nature of the modern smartphone, and security practices have been completely rethought to deal with the challenge of alien, uncontrolled devices being brought inside the corporate firewall. Many enterprises have dealt with this issue by requiring employee devices to adhere to certain security policies. Yet, this in turn has raised some privacy issues for the end users who might sacrifice their personal data to comply with such policies.

Controlling these concerns from different perspectives has proven to be just too difficult for both the company and the user. Fortunately, with the option to tap into virtualization solution, a user can differentiate between these contexts through virtual phone dedicated for each. Similar to virtualization on servers and desktop machines, mobile virtualization allows several virtual phones (VPs) to run simultaneously on the same physical mobile device with a clear isolation among them. Leveraging mobile virtualization, dual persona devices have been already released, enabling two phones-in-a-phone, to support BYOD needs. Yet, satisfying many of our daily-life needs nowadays, urges mobile devices to broaden their capabilities to support more than just two contexts and drive multi-persona device to be the new game changer. Multi-persona creates the same impenetrable wall between an employee's applications and an organization's data and applications, yet allowing one phone to co-host "multiple completely independent and secure virtual environments". But why would anybody want more than just two virtual phones? With multi-persona, a user could isolate private banking services and e-commerce, sensitive corporate data, social networking and games in separate VPs, in order to efficiently manage financial transactions, prevent untrusted applications from accessing critical information and share the device with other family members without ending up with accidental phone calls, unintended in-app purchases or even access to restricted content. Even more interesting use cases for multi-persona come with having multiple work VPs with different levels of security. While working at their private clinic and at multiple hospitals, doctors are subject to different mobile policies, reflecting each of the different institutions. With personal, clinic and hospitals VPs, multi-persona allow doctors to comply with the policy of each and effectively treat their patients while maintaining their own unburdened personal use of the device. As a matter of fact, the success of multi-persona lies in its capability of supporting multiple virtual devices concurrently on a single terminal, while making the latter able to clearly distinguish between the different contexts in which it is used.

Though its isolation competency, mobile virtualization imposes significant overhead on the mobile terminal with limited computation capabilities, memory capacity and battery lifetime. In one hand, previous works [Tout *et al.* (2015, 2016)] have proved that even a lightweight

virtualization architecture can be costly for the mobile device resources causing performance degradation and more critically shorter viability of the system. On the other hand, technological advances have markedly shaped the way computations are performed. With the abundance of cloud resources, many computation offloading approaches have been proposed to support mobile devices needs by migrating the computations from an end mobile device to remote resourceful infrastructure. All these approaches have indeed proved their competency to reduce the resource consumption on the mobile terminals and enhance the latter performance [Zhou *et al.* (2016); Hung *et al.* (2012); Cuervo *et al.* (2010); Kosta *et al.* (2012); Kemp (2014); Chen *et al.* (2012); Chun *et al.* (2011); Shi *et al.* (2014); Chae *et al.* (2014); Gordon *et al.* (2012); Flores *et al.* (2014); Xia *et al.* (2014)]. From these premises, we proposed in previous work offloading-based approach to support multi-persona [Tout *et al.* (2016)]. Through optimization and heuristics, the solution was capable to find the dissemination of computations, in each virtual phone, between local and remote execution capable of minimizing the resource consumption and augmenting the applications' performance. However, in many scenarios, the capability of only offloading computations might just not be sufficient to manage the device resources.

"Why offloading might not be enough as a resource management solution?" Starting with the first scenario (Sc_1) where typically some computations are not offloadable whether based on their type (native tasks), security level, or even their need to call device-related functionalities (e.g., camera); when more resources are needed on the device, offloading these components is not an option and different management decision should be taken. In another scenario (Sc_2), the device can run out of resources (e.g., battery) while some applications have been idle for a period of time and will not be used in the near future, yet still consuming part of the device resources. Offloading such applications will not be efficient as it might cost the device more energy for transmitting the needed data for remote processing. Moreover, imagine the device is running out of resources while some VPs are idle (i,e., either running idle applications or none) and will not get active in the near future; also here just an offloading decision will not be able to efficiently cope with such situation as the virtual phones are still stressing out the device

resources (Sc_3). Further, typically, local and remote application execution are adopted based on their capability to enhance the performance of an application or reduce the energy consumed on the device. However, in some use cases, one of these strategies should be enforced independently of the cost it imposes on the applications performance. For instance, the case when the device is running out of battery, while local execution of the running components outperforms their remote execution (Sc_4).

"So what kind of advanced manageability is effectively needed besides offloading?" Along with offloading, other manageability solutions should be available. Shutting down, whether idle applications which will not be used in the next period of time or active non offloadable components can cope with the first two aforementioned scenarios (Sc_1, Sc_2). While the ability to switch off particular virtual phones can release the device resources and hence cope with (Sc_3). Finally, prioritizing system viability over individual app performance when evaluating offloading cost is critically needed in (Sc_4) as the viability of the whole system is more critical than the performance of particular application(s).

"How does this work address these research problems and offer the needed manageability solutions?" We propose in this work a proactive solution with advanced manageability to control the virtual instances running on the mobile device. Through machine learning intelligence [Kodratoff (2014)], our proposition is able to predict beforehand the context and the resource needs for future execution and manage the device resources accordingly. It is able to predict the resource needs of the currently running VPs and applications as well as those VPs and applications that may run in the future and their resource demands, all based on the usage pattern of the device. Additionally, the solution offers advanced management strategies, particularly, offloading the execution, turning off components, switching off personas and prioritizing device survivability over applications performance, all according to the contextual environment that might affect the application of these strategies. Based on the predicted context and resource needs, detected hotspot (active and resource intensive) and coldspot (idle) components and virtual phones, classification scheme, network conditions and device state, an optimization model is formulated. We propose respectively a dynamic programming algorithm

to solve this model, where the solution indicates for each computation the adequate strategy to handle the resource and performance needs on the end terminal. We performed thorough experiments and the results prove the efficiency of our proposition to manage VPs needs.

This work offers the following contributions:

- Provide proactive machine learning technique to predict future context and resource needs.
- Propose advanced strategies to manage VPs.
- Build a novel optimization model to meet with the resource needs and enhance the performance on end devices with multiple VPs.
- Present efficient dynamic programming algorithm to find the adequate strategies to be applied by the end terminal.

5.3 Related Works

We survey in this section predictive strategies for virtual machines management, recent techniques for mobile computations offloading, and algorithms for dynamic offloading decisions.

5.3.1 Predictive Virtual Instances Management Strategies

Sharing an end terminal between several virtual machines raises many problems and autonomic load balancing of resources is one of these open key challenges to be resolved. In this context, different approaches have been proposed for load prediction on a physical machine. Predicting future load enables proactive consolidation of VMs on the overloaded and under-loaded physical machines [Farahnakian *et al.* (2015)]. In [Farahnakian *et al.* (2013a)] and [Farahnakian *et al.* (2013b)], the authors have proposed regression methods to predict CPU utilization of a physical machine. These methods use the linear regression and the K-nearest neighbor (KNN) regression algorithms, respectively, to approximate a function based on the data collected during the lifetimes of the VMs. The formulated function is then used to predict an overloaded

or an under-loaded machine. A linear regression based approach has been implemented by Fahimeh Farahnakian [Farahnakian *et al.* (2013a)]. The CPU usage of the host machine is predicted on the basis of linear regression technique and then live migration process was used to detect under-utilized and over-utilized machines. Bala et al. [Bala & Chana (2016)] have proposed a proactive load balancing approach that based on prior knowledge of the resource utilization parameters, applies machine learning techniques to predict future resource needs. Various techniques have been studied in their work, such as KNN, ANN, SVM and RF and the one with maximum accuracy has been utilized as prediction-based approach. Xiao et al. [Xiao *et al.* (2013)] have also used a load prediction algorithm to capture the rising trend of resource usage patterns and help identifying hot spots and cold spots machines. After predicting the resource needs, Hot spot and cold spot machines are identified. When the resource utilization of any physical machine is above the hot threshold, the latter is marked as hotspot. If so, some VMs running on it will be migrated away to reduce its load [Xiao *et al.* (2013)]. On the other hand, cold spot machines either idle or having the average utilization below particular threshold, are also identified. If so, some of those physical machines could potentially be turned off to save energy [Xiao *et al.* (2013)] [Beloglazov & Buyya (2010)].

5.3.2 Dynamic Offloading Algorithms

A Dynamic Programming (DP) algorithm was proposed in [Liu & Lee (2014)], where a two dimensional DP table was used in order to determine what to offload. However, a backtracking algorithm was needed to find the final decisions, which was time consuming. A dynamic offloading algorithm based on Lyapunov optimization was presented in [Huang *et al.* (2012)]. The algorithm is based upon a relationship between the current solution and the optimal solution requiring a considerable amount of execution time and many iterations to converge upon a solution [Shahzad & Szymanski (2016)]. Another dynamic programming approach has been proposed in the same context [Toma & Chen (2013)], yet it doesn't consider the energy consumed in the mobile device which is an important criteria for resource constrained mobile devices. A semidefinite relaxation approach for the offloading problem was presented by Chen

et al., [Chen *et al.* (2015a)]. Their work considered a mobile cloud computing scenario consisting of one nearby computing access point, and a remote cloud server(s). Their proposition is based on an algorithm that solves a linear program through randomization and relaxation to generate an integer solution. The algorithm can find a near-optimal solution when using about 100 trials of relaxation. In [Shahzad & Szymanski (2016)], a dynamic programming algorithm called DPH is proposed. The algorithm generates periodically random bit strings of 0s and 1s, for remote and local execution of tasks, and utilize sub-strings when they improve the solution. The algorithm can find a nearly optimal solution after several iterations. The authors use a Hamming distance criterion to terminate the search process and hence obtain the final decision quickly. The stopping criterion is met when a given fraction of tasks are offloaded.

5.3.3 Our Contributions

Thinking about the multiple VPs case as the load balancing problem of virtual machines in the cloud, we propose in this work an analogous proactive solution with advanced manageability strategies. Our proposition includes first a prediction-based proactive approach using machine learning techniques, which have been found suitable from the literature review discussed above. As pro-activity requires a prior knowledge of the device context and resource utilization, we devote a set of profilers to log the relevant data. Gathered data are trained using machine learning algorithms and predictions about future context and resource needs are made. Various machine learning techniques are evaluated, namely, Linear Regression (LR) [Seber & Lee (2012)], Support Vector Regression (SVR) [Drucker *et al.* (1997)], Neural Network (NN) [Demuth *et al.* (2014)] and Deep Neural Network (DNN) [Goodfellow *et al.* (2016)], and the one found to be with highest accuracy (SVR) has been utilized as prediction-based technique. Moreover, advanced management solution is proposed, which includes different strategies; namely, offloading and turning off tasks and switching off personas. To help adopting such strategies, we propose hotposts and coldspots detectors along with a classification scheme to categorize and prioritize tasks and VPs. Based on the predicted context and resource needs, identified hotspot and coldspot tasks, prioritization, device state and network connection properties, a

multi-objective optimization model is formulated. Finally, the model is solved with a dynamic programming algorithm that finds the adequate strategies to be applied for each task and VP aiming to attain the objective functions in the formulated model. The experimental results clearly demonstrate the effectiveness of this work, showing better resource management solution with improved performance.

5.4 System Model

The capacity of the physical device should be sufficient to satisfy the resource needs of all VPs it is hosting. Otherwise, the physical mobile device is overloaded and will lead to degraded performance of its VPs or even system crash when there is no enough resources to support future VPs needs. Besides the offloading strategy offered in this work, some green computing actions can be taken; VPs and applications can be switched off to save resources and ensure longer viability on the physical device as long as they are idle and won't be used in the near future or whenever the whole system is in critical situation. The proposed system model is depicted in Figure 5.1.

A set of profilers are installed in each VP to monitor different aspects. The program profiler monitors tasks' energy consumption, execution time and size of data to be transmitted in case of remote processing. The network profiler monitors the network characteristics in terms of availability, type (e.g., wifi, 3G), bandwidth, latency and energy consumed on transmission. The device profiler inspects the energy consumption on the device as well as the battery level, CPU utilization, memory consumption, and keep track of the location and time where each VP and application is used. The gathered information helps predicting future usage context and resource needs.

The statistics collected at each VP are forwarded to the predictor component, which applies machine learning techniques to predict the context of the device in the future time slot. The context includes timestamp, location, VPs running, components running and resource needs accordingly. In other words, this includes the resource needs of the currently running VPs and

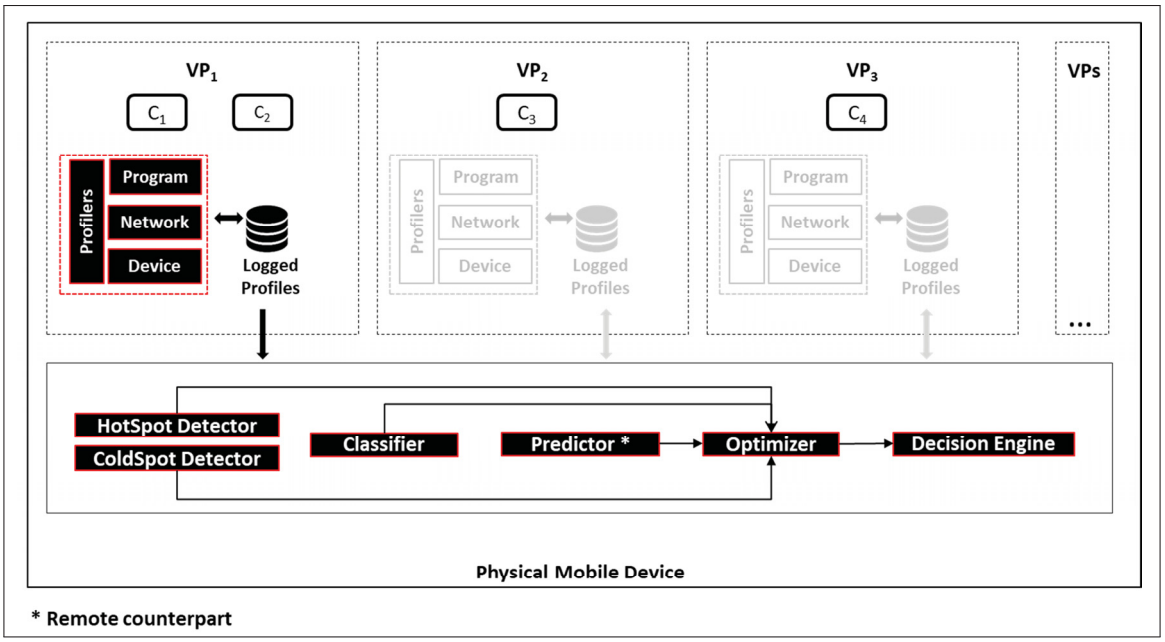


Figure 5.1 System model.

applications as well as those will run in the future and their resource demands. The predictor aims to provide proactive solution capable of avoiding critical situations of performance degradation or system crash. Any of the local or remote predictors can be invoked to train the model and perform the prediction. The performed experiments presented later in this work investigates remote prediction while the evaluation of processing training and prediction locally and their effect on the device accordingly are to be investigated in future work.

To manage limited system resources, the Android system can terminate running applications. If the Android system needs to free up resources and terminate processes, it uses the following priority system [Android (2016)]. Based on its status, each process is assigned a priority level. Foreground, visible, service, background and empty processes are assigned priorities from 1 to 5 respectively. Those processes with higher priority levels have higher chance to be terminated. To turn off particular applications and VPs we apply the following classification scheme for prioritization. We define 5 classes to categorize a **component**

- Offloadable/Notoffloadable

- Background/Foreground
- Active/Idle: doesn't actively utilize system resources
- Belong to actual context VP (AC)/Not (DC)
- Critical/Not: based on user preferences

As for **VPs**, we define:

- Background/Foreground
- Active/Idle: is not running any component which actively utilize system resources
- Define actual context (AC)/Not (DC)
- Critical/Not: based on user preferences

Table 5.1 shows the prioritization scheme applied.

Table 5.1 Prioritization scheme.

Class				Priority
Critical	AC	Active	Foreground	1
Critical	AC	Idle	Foreground	2
Critical	AC	Active	Background	3
Critical	AC	Idle	Background	4
Critical	DC	Active	Foreground	5
Critical	DC	Idle	Foreground	6
Critical	DC	Active	Background	7
Critical	DC	Idle	Background	8
Uncritical	AC	Active	Foreground	9
Uncritical	AC	Idle	Foreground	10
Uncritical	AC	Active	Background	11
Uncritical	AC	Idle	Background	12
Uncritical	DC	Active	Foreground	13
Uncritical	DC	Idle	Foreground	14
Uncritical	DC	Active	Background	15
Uncritical	DC	Idle	Background	16

Frequently invoked, resource-intensive and/or time consuming components are designated as hotspots. Such criteria and the thresholds used for hotspot selection are adjusted automatically based on the device state, instrumented by the profilers. The hotspot detector analyzes the logged profile information and collects all the components whose processing, memory, data size, execution time and frequency of call are greater than given threshold values respectively. These components are marked as hotspots. Higher thresholds will keep the hotter components. For more information about the hotspot detector and its algorithm, please refer to [Tout *et al.* (2017)]. In contrast, the coldspot detector detects idle components and VPs. An idle VP is one that is not running any active component or just idle ones.

According to the predicted context and resource needs, classification scheme, hotposts, colspots, and instrumented data a multi-objective optimization problem of three vectors of variables is formulated in the optimizer module. The objective functions aim to manage the resource needs of VPs, avoid performance degradation and system crash. Finally, for the decision engine, we propose a novel algorithm based on dynamic programming to find the adequate strategies to be applied by the physical device. The decision implies which VPs should remain on and/or those to be switched off, and for each component wether it should be powered-off, executed locally or offloaded.

5.5 Machine Learning Prediction

To predict future context and usage behavior, we study a variety of machine learning techniques [Kodratoff (2014)], namely, Linear Regression (LR) [Seber & Lee (2012)], Support Vector Regression (SVR) [Drucker *et al.* (1997)], Neural Network (NN) [Demuth *et al.* (2014)] and Deep Neural Network (DNN) [Goodfellow *et al.* (2016)]. We compare the accuracy of these techniques and the one found to be with the highest accuracy is to be used throughout this work.

5.5.1 Linear Regression

Linear regression [Seber & Lee (2012)] is a linear model that assumes a linear relationship between the input variable(s) x and the single output variable y. In other words, y can be calculated from a linear combination of the input variables x. The method is referred to as simple linear regression, when there is one input variable and as multiple linear regression method when there are multiple input variables.

The linear equation assigns one scale factor to each input, called a coefficient (β). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot), often called the bias coefficient or intercept (β_0). In a simple regression problem, the form of the model would be: $y = \beta_0 + \beta_1 * x$

In higher dimensions when more than one input variable x are used, the line is called a plane or a hyper-plane and the linear relationship can be expressed as: $y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n$

5.5.2 Support Vector Regression

Support vector machine (SVM) analysis is a popular machine learning tool for classification and regression (SVR) [Drucker *et al.* (1997)]. SVR is considered a nonparametric technique because it relies on kernel functions. The model generated by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

The objective is to

Minimize $\frac{1}{2}||w||$

Subject to $\begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases}$

where x_i is a training sample with target value y_i. The inner product plus intercept $\langle w, x_i \rangle + b$ is the prediction for that sample, and ε is a free parameter that serves as a threshold; all predictions have to be within an ε range of the true predictions.

5.5.3 Neural Network

Neural Network (NN) [Demuth *et al.* (2014)] also called Artificial Neural Network (ANN), is a learning algorithm inspired by the structure and functional aspects of biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

5.5.4 Deep Neural Network

A deep neural network (DNN) [Goodfellow *et al.* (2016)] is an ANN with multiple hidden layers between the input and output layers. Similar to ANN, DNN can model complex nonlinear relationships. DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back.

5.6 Problem Formulation

In the following, we mathematically formulate the Multiple Virtual Phones (MVPs) problem as a multi-objective optimization model with three vectors decision variables.

- Decision Variables:

  $I_{vp_j} = \{I_{vp_1}, ...I_{vp_m}\}$

  $I_{c_i,vp_j} = \{I_{c_1,vp_1}, ...I_{c_n,vp_m}\}$

$$x_{c_i,vp_j} = \{x_{c_1,vp_1}, ...x_{c_n,vp_m}\}$$

where,

$$\forall vp_{j,j:1\to m}, I_{vp_j} = \begin{cases} 0, & \text{if } vp_j \text{ should be switched off} \\ 1, & \text{if } vp_j \text{ should remain active} \end{cases}$$

$$\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}, I_{c_i,vp_j} = \begin{cases} 0, & \text{if } c_i \in vp_j \text{ should be turned off} \\ 1, & \text{if } c_i \in vp_j \text{ should remain on} \end{cases}$$

$$\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}, x_{c_i,vp_j} = \begin{cases} 0, & \text{if } c_i \in vp_j \text{ is to be executed locally} \\ 1, & \text{if } c_i \in vp_j \text{ is to be offloaded} \\ -(indifferent), & \text{if } I_{c_i,vp_j} = 0 \end{cases}$$

- Notations:

m	number of virtual phones
n	number of components in a virtual phone
j	1,... m
i	1,... n
vp_j	virtual phone
c_i	component
p_{cpu}	power consumed by vp_j on precessing
p_s	power consumed by vp_j on the screen
p_{cpu}^{idle}	power consumed by vp_j on idle CPU
p_{net}^{active}	power consumed on network being active
p_{tr}	power consumed for data transmition
$Data_{c_i,vp_j}$	size of data transmitted for offloading c_i, $c_i \in vp_j$
CPU_{c_i,vp_j}^{local}	cpu usage in vp_j by c_i executed locally
CPU_{c_i,vp_j}^{remote}	cpu usage in vp_j by c_i offloaded
$Memory_{c_i,vp_j}^{local}$	memory usage in vp_j by c_i executed locally
$Memory_{c_i,vp_j}^{remote}$	memory usage in vp_j by c_i offloaded
t_{c_i,vp_j}^{local}	time to execute c_i locally, $c_i \in vp_j$
t_{c_i,vp_j}^{remote}	round trip time to process c_i remotely, $c_i \in vp_j$
W_{cpu}	weight for objective function (5.1)
W_{memory}	weight for objective function (5.2)
W_{energy}	weight for objective function (5.3)
W_{time}	weight for objective function (5.4)

$\tilde{T}_{cpu}$	threshold for cpu usage
$\tilde{T}_{memory}$	threshold for memory usage

- Mathematical Model:

$$F = Minimize \begin{bmatrix} F_{cpu} \\ F_{memory} \\ F_{energy} \\ F_{time} \end{bmatrix}$$

subject to

$$\begin{aligned} F_{cpu} &< \tilde{T}_{cpu} && (c_1) \\ F_{memory} &< \tilde{T}_{memory} && (c_2) \end{aligned}$$

where,

$$F_{cpu}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{cpu} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times CPU_{c_i,vp_j}^{local} + \sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times CPU_{c_i,vp_j}^{remote}\right] \tag{5.1}$$

$$F_{memory}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{memory} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times M_{c_i,vp_j}^{local} + \sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times M_{c_i,vp_j}^{remote}\right] \tag{5.2}$$

$$F_{energy}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{energy} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times \left(\left(P_{cpu} + P_s\right) \times t_{c_i,vp_j}^{local}\right) + \sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times \left(\left(P_{cpu}^{idle} + P_s + P_{net}^{active}\right) \times t_{c_i,vp_j}^{remote} + \left(P_{tr} \times \left(Latency + \frac{Data_{c_i,vp_j}}{Bandwidth}\right)\right)\right)\right] \tag{5.3}$$

$$F_{time}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{time} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times t_{c_i,vp_j}^{local} + \sum_{j=1}^{m} I_{c_i,vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times \left(t_{c_i,vp_j}^{remote} + Latency + \frac{Data_{c_i,vp_j}}{Bandwidth}\right)\right] \tag{5.4}$$

The aim of this model is to find the strategies (determined by the decision variables I_{vp_j}, I_{c_i,vp_j} and x_{c_i,vp_j}) able to minimize the resource usage on the physical mobile device in order to avoid

performance degradation, while assuring the availability of predicted future context and resource needs (identified in constraints (c_1) and (c_2)). Equation (5.1) measures the processing needed to execute services locally on the device and the one needed waiting for remote computations and/or processing the response back on the terminal. Equation (5.2) calculates the memory needed to process local computations and the one consumed while waiting and/or processing the response back. Equation (5.3) is used to determine the energy consumed by the CPU and on the screen brightness for for local processing as well as the energy spent on idle CPU, screen brightness and active network while waiting the execution of offloaded services. It also includes the energy consumed by the terminal for data transmission. Finally, Equation (5.4) calculates the duration of local and remote processing taking into account the latency for data transmission.

5.7 Proposed Dynamic Programming Algorithm

Dynamic programming (DP) [Bertsekas *et al.* (1995)] is a powerful technique that can be used to solve many problems, including optimization, for which a naive approach would take exponential time to finish the process. We present in this section our proposed dynamic programming algorithm, which aims to find the strategy to be applied for the VPs and their components, while meeting with the objective functions defined in previous section.

5.7.1 DP Table Filling

Since at the lower level, strategies should be determined for each service in the running personas, we need to generate a bit stream of N bits, where N is the number of components on the device. We use an $N+1*N+1$ DP table to store the bit-streams showing which personas are to be switched off and which components to be offloaded, executed locally and those to be turned off. For the first step, a random bit stream of size N*3(number of decision variables) is generated that determines a first solution. However, different **Rules apply when generating potential random solutions**:

- $\forall c_{i,i:1\rightarrow n}, \forall vp_{j,j:1\rightarrow m}$, If $I_{vp_j} = 0$ then $I_{c_i,vp_j} = 0$ and $x_{c_i,vp_j} = -$
- $\forall c_{i,i:1\rightarrow n}, \forall vp_{j,j:1\rightarrow m}$, If $I_{vp_j} = 1$ and $I_{c_i,vp_j} = 0$ then $x_{c_i,vp_j} = -$
- $\forall c_{i,i:1\rightarrow n}, \forall vp_{j,j:1\rightarrow m}$, If $I_{vp_j} = 1$ and $I_{c_i,vp_j} = 1$ then $x_{c_i,vp_j} = 0$ or 1(if c_i is offloadable)
- $\forall c_{i,i:1\rightarrow n}, \forall vp_{j,j:1\rightarrow m}$, I_{vp_i} is constant $\forall c_i \in vp_j$
- Based on the prioritization scheme in Table 5.1, the probability of generating a 0/1 bit is adapted.
- Based on wether c_i is offloadable or not, x_{c_i,vp_j} is generated.

Rules to fill the DP Table: This stream is assigned to the table such that 1s for x_{c_i,vp_j} are assigned to the next horizontal cell, and the 0s are assigned to the next vertical cell. If x_{c_i,vp_j} in the stream is 1, the starting cell is $(1,2)$ and if it is $0/-$, the starting cell is $(2,1)$. This approach will avoid extra computations for common bit strings [Xiao *et al.* (2013)]. Since the first cell is left empty, and the stream size is N, we need $N+1*N+1$ Table to fit the generated streams.

Example: A $2D$ $6*6$ table is shown in Table 5.2. To clarify, assume that $N = 5$ (2 components in vp_1 and 3 components in vp_2) and the first random stream is 00-00-10-110111 (non bold vectors). Assume that the second random bit stream is 11111010-10-110. The starting cell of the second stream is $(1,2)$ since the third bit is 1. By following the aforementioned rules to fill the table, the resulting bold stream is shown in the table. Whenever a bit stream is generated

Table 5.2 DP Table Filling

	[1,1,1]				
[0,0,-]	**[1,1,0]**				
[0,0,-]	**[1,0,-]**				
[1,0,-]	**[1,0,-]**				
[1,1,0]	[1,1,1]/**[1,1,0]**				

randomly, we calculate the consumed CPU, memory, energy and time of each cell (i.e., each component) in the table, and also at the same time calculate the total of each of these metrics of this bit stream, which formulate the defined objective functions. However, if a random bit stream is generated which has some common cells with an existing string in the table; we replace that cell with the new value only if it's able to offer better trade-off with respect to the defined objective functions, for that cell, conforming with the weight of each metric. We then

update the metrics of the remaining cells for the existing bit streams, based on the new values at this common cell. Every time a new stream is generated, we keep tracking the arrangement of the stream in Table 5.2.

When to terminate this process? Once a solution that meets with the device needs is generated with the least lost possible (i.e, least number of switched off personas and components). Therefore, we define the latter as the hamming distance between the I_{vp_j} and I_{c_i,vp_j} in the generated bit stream solution and $I_{vp_j} = 1$ and $I_{ci,vp_j} = 1$.

The full process described above is depicted in Algorithm 5.1.

Algorithm 5.1 Dynamic Programming Algorithm

```
do
    Set the resource constraints based on the device state and
    predicted future context
    Initialize T_bitstreams
    Generate a random bit stream that conforms with the rules
    Check the first bit to specify the starting cell in the table
    for i = 1 to N do
        Put each bit of the bit stream in the correct position in table
        Calculate the self-CPU, memory, energy and time of each cell
        and their corresponding totals (objective functions)
        if this specific cell in table is visited before then
            Compare the new self-CPU, memory, energy and time of
            this cell with the previous one
            if the new values of the cell offer better trade-off than
            the previous one   then
                Replace the values of this cell with the new calculated
                amounts
                Update the remaining amounts in the remaining cells
                of the previous bit stream based on the new amount
                of this common cell
                Calculate the cpu, memory, energy and time of the
                remaining bits of the new bit stream
                Track the position of all bits in the table in a matrix
            else
                Keep the previous totals for the cell
                Track the position of all bits in the table in a matrix
            end if
        end if
    end for
    return bit stream with least loss and its corresponding F_cpu,
        F_memory, F_energy, F_time
while No feasible stream is found &NF_bitstreams < T_bitstreams
```

5.8 Evaluation

In the sequel, we first evaluate the accuracy of the machine learning techniques discussed in Section 5.5 and the one with the least error rate is adopted. According to the future context, predicted by the latter, we evaluate the efficiency of our proposed algorithm to find the adequate strategies that meet with the optimization objectives and comply with the future resource needs.

5.8.1 Setup

First, we evaluate LR, SVR, NN and DNN in two phases context-aware prediction:

Phase #1: Predict resource needs (CPU and Memory) for the running personas/apps

T_0: predict resource needs at T_1 for currently running personas/apps.

Phase #2: Predict future running/switched off personas/apps and their resource needs (CPU and Memory)

T_0: predict behavior (Location, VPs and Components) at T_1
T_0: then resource needs accordingly

For phase #1, we generate two data sets (DS_1 and (DS_2)) for doctor work shifts during one day from 8 am to 8 pm. Both data sets have 14400 rows with data being generated each 3 seconds. The dataset consists of timestamp, which allows daily pattern recognition, coordinates to detect the location of the user (e.g., home, clinic, hospitals), number of VPs running, number of components, their CPU usage and memory consumption. The data set disregards the time spent to move from one location to another. The usage behavior in this data set is described in Table 5.3. DS_1 reflects normal behavior while DS_2 includes some peaks in the resource consumption to see how would that affect the prediction accuracy of these resources.

Table 5.3 Device Usage Behavior

Time	# of VPs	VP(s)	Location	# of Components
8:00-10:00	1	Personal	Home	1-5
10:00-12:00	2	Personal + Clinic	Clinic	3-8
12:00-15:00	2	Personal + Hospital#1	Hospital#1	3-8
15:00-18:00	3	Personal + Hospital#1 + Hospital#2	Hospital#2	5-12
18:00-20:00	1	Personal	Home	1-5

As for phase #2, we generate 4 datasets (DS_3, DS_4, DS_5 and DS_6) for daily prediction. Data is generated every 30 min to reduce the data size to be analyzed. Future Context, behavior and resource needs are all to be predicted in this phase. In DS_3, the same usage pattern is shown daily (day 1, 2, 3 and 4 and prediction applied on day 5) In DS_4, same usage pattern is shown daily (day 1, 2 and 4, Spikes on day 3 and prediction applied on day 5). In DS_5, similar daily usage pattern (day 1, 2, 3 and 4 and prediction applied on day 5). In DS_6, similar daily usage pattern (day 1, 2 and 4, Spikes on day 3 and prediction applied on day 5).

Table 5.4 ML techniques setup for phase #1

LR		**SVR**		**NN**		**DNN**	
Window size	1h	**Kernel**	Radial	**Hidden layers**	1	**Hidden layers**	2
Train	50 min	**Cross-validation**	10	**Nodes**	5	**Nodes**	5
Test	10 min	**Cost**	300	**Window size**	1h	**Threshold**	0.01
		Window size	1h	**Train**	50 min	**Window size**	1h
		Train	50 min	**Test**	10 min	**Train**	50 min
		Test	10 min			**Test**	10 min

Table 5.5 ML techniques setup for phase #2

LR		**SVR**		**NN**		**DNN**	
Window size	4 days	**Kernel**	Radial	**Hidden layers**	1	**Hidden layers**	2
Train	4 days	**Cross-validation**	10	**Nodes**	4-7	**Nodes**	4-7
Test	1 day	**Cost**	300	**Window size**	4 days	**Threshold**	0.01
		Window size	4 days	**Train**	4 days	**Window size**	4 days
		Train	4 days	**Test**	1 day	**Train**	4 days
		Test	1 day			**Test**	1 day

Various machine learning techniques are used and compared: LR=Linear regression model, SVR=support vector regression, NN=neural network and DNN=Deep Neural Network. Tables 5.4 and 5.5 show the setup of each machine learning technique studied in each prediction phase.

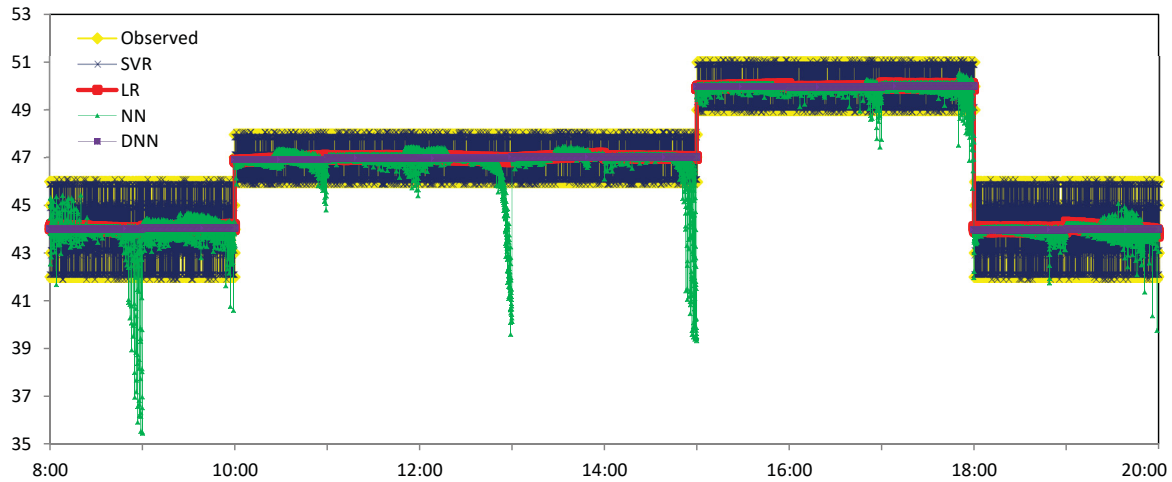
5.8.2 Numerical Analysis

For each dataset in both phases #1 and #2, we examine the accuracy of Liner Regression (LR), Support Vector Regression (SVR), Neural Network (NN) and Deep Neural Network (DNN) techniques to predict resource needs in terms of CPU and memory, to define the constraints of the formulated optimization model.

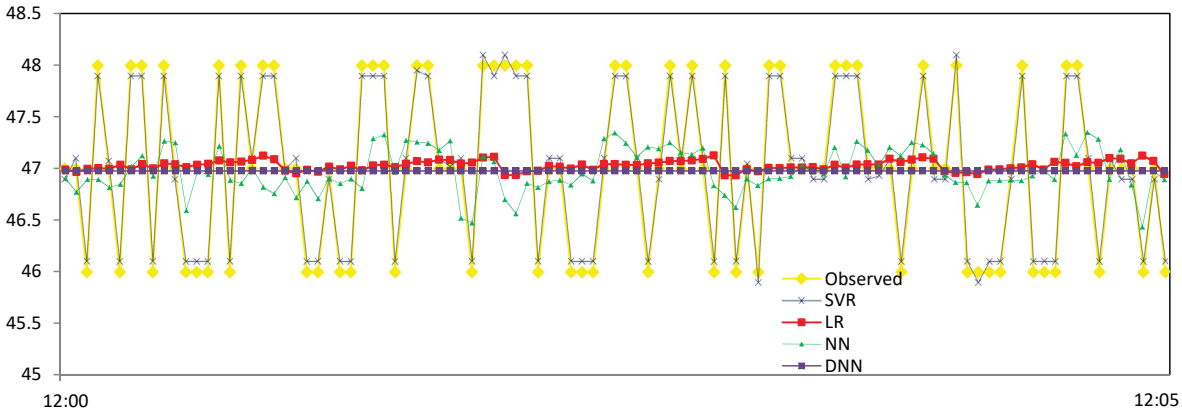
Figure 5.2 shows the results of Dataset DS_1. Particularly, Figure 5.2a depicts the CPU values over the day from 8:00am till 8:00pm, while in Figure 5.2b, only a subset of these values is illustrated for the convenience of the reader when examining the accuracy of each technique. *Observed* values are the real data. Comparing the latter with those predicted by each machine learning technique, the results show that SVR outperforms the others with the highest accuracy (SVR predicted values almost matches the Observed values). In Figure 5.2c, we analyze the error rate of these predictions based on the Root Mean Square Error metric (RMSE). SVR shows the minimum RMSE among the other techniques with only 0.098 error rate. The same analysis applies on the memory results depicted in Figure 5.3. SVR shows the highest accuracy with 0.0982 RMSE.

For DS_2, which includes some unusual behavior (peaks) in the resources consumption, Support Vector Regression (SVR) outperformed the other techniques as well for both CPU and memory needs prediction, showing the least error rate with 0.098083 and 0.09825 RMSE accordingly as depicted in Figures 5.4 and 5.5.

$DS_3, DS_4, DS_5 and DS_6$, all imply multi-stage prediction, which includes prediction of the location followed by the virtual phones and components and finally the resource needs by the latter.



a) CPU values over the day (8:00-20:00)



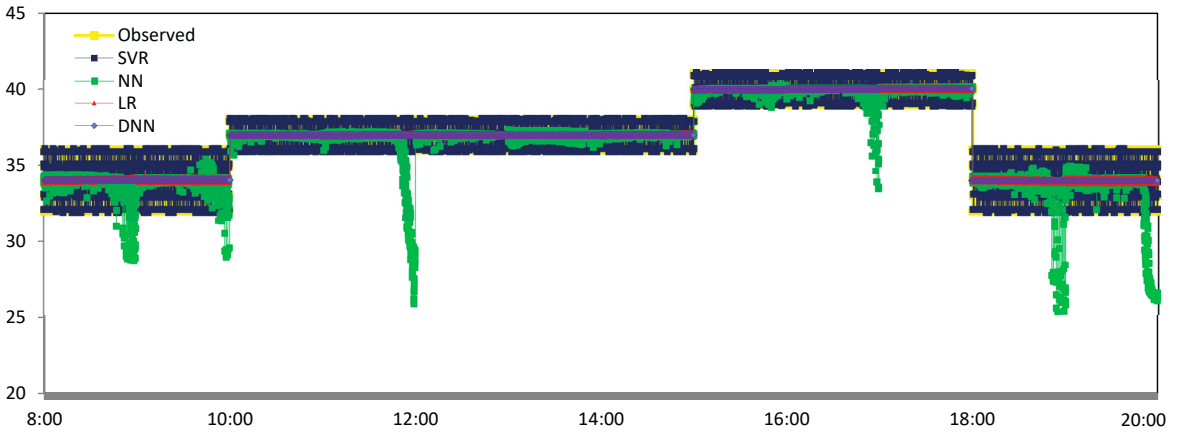
b) Subset of CPU values (12:00-12:05)

RMSE-CPU			
LR	SVR	NN	DNN
1.008667	0.098083	1.331917	1.020627

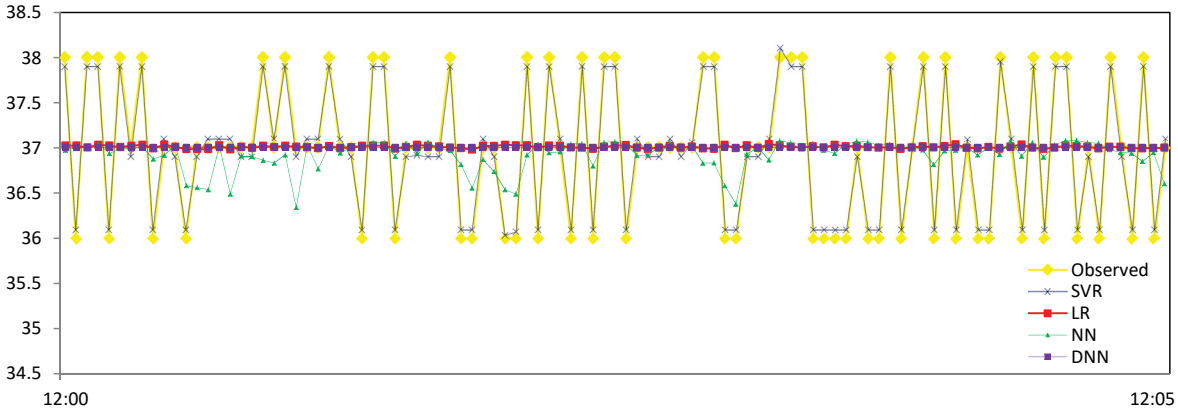
c) RMSE

Figure 5.2 DS1: observed CPU vs. predicted.

When exactly the same context and usage pattern are repeated from day 1 to day 4, SVR was able to predict that the same applies on the fifth day with no errors detected in the predicted CPU and memory values as depicted in Figure 5.6. However, when exactly the same usage pattern is shown daily (day 1, 2 and 4) with some spikes observed on day 3, the error rate for all the machine learning techniques has increased for the predicted values on day 5 as depicted in Figure 5.7. In this case, LR, SVR, NN and DNN have showed error rates of 3.62, 1.79, 2.11



a) Memory values over the day (8:00-20:00)



b) Subset of memory values (12:00-12:05)

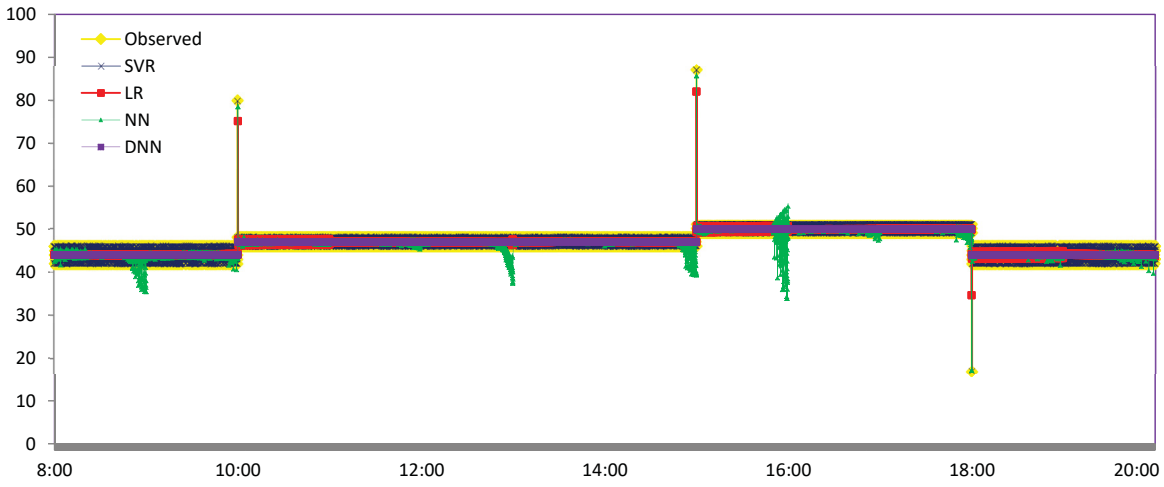
RMSE-Memory			
LR	SVR	NN	DNN
1.015917	0.09825	1.623208	1.021837

c) RMSE

Figure 5.3 DS1: observed memory vs. predicted.

and 3.26 respectively for the CPU values and 3.91, 1.65, 1.44 and 3.37 RMSE for the memory values, while higher accuracy was observed in previous case (DS_3) with 2.37, 0, 1.89 and 2.38 RMSE respectively for CPU and 2.47, 0, 1.34 and 2.52 RMSE for the memory values.

The same analysis applies when comparing the results in Figures 5.8 and 5.9, where similar, but not exactly the same, context and usage behavior are observed on daily basis for DS_5 and some spikes injected in DS_6.

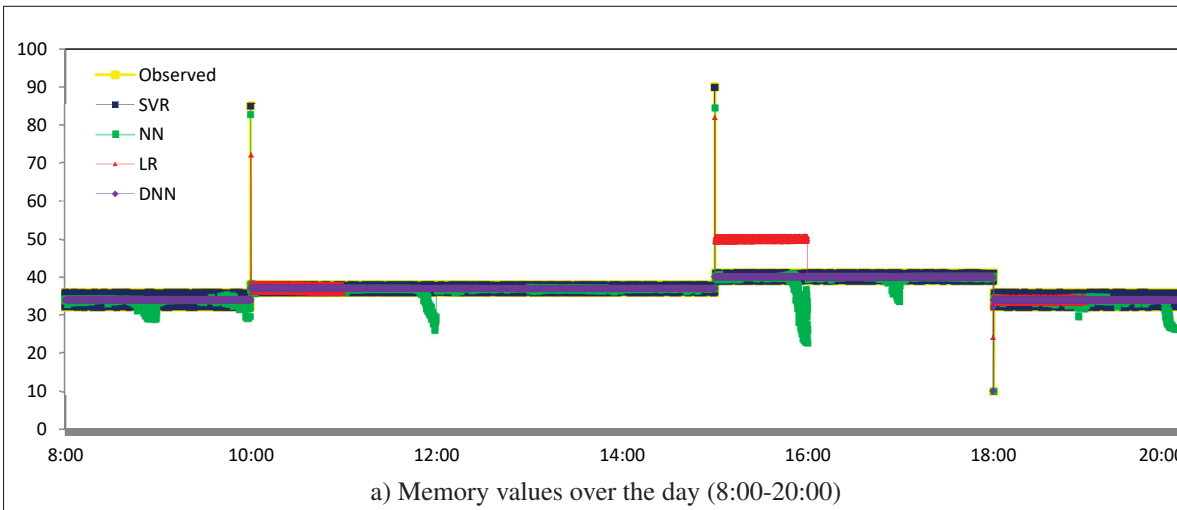


a) CPU values over the day (8:00-20:00)

RMSE-CPU			
LR	SVR	NN	DNN
1.008667	0.098083	1.331917	1.020627

b) RMSE

Figure 5.4 DS2: observed CPU vs. predicted.

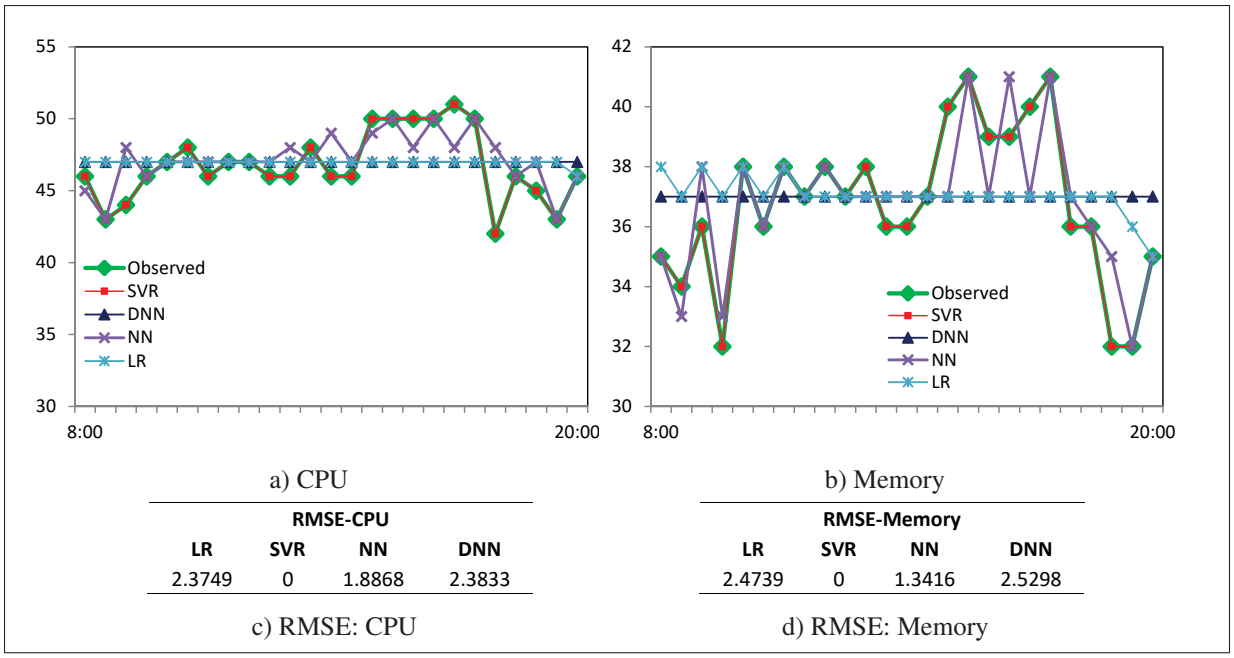


a) Memory values over the day (8:00-20:00)

RMSE-Memory			
LR	SVR	NN	DNN
1.015917	0.09825	1.623208	1.021837

b) RMSE

Figure 5.5 DS2: observed memory vs. predicted.



a) CPU

b) Memory

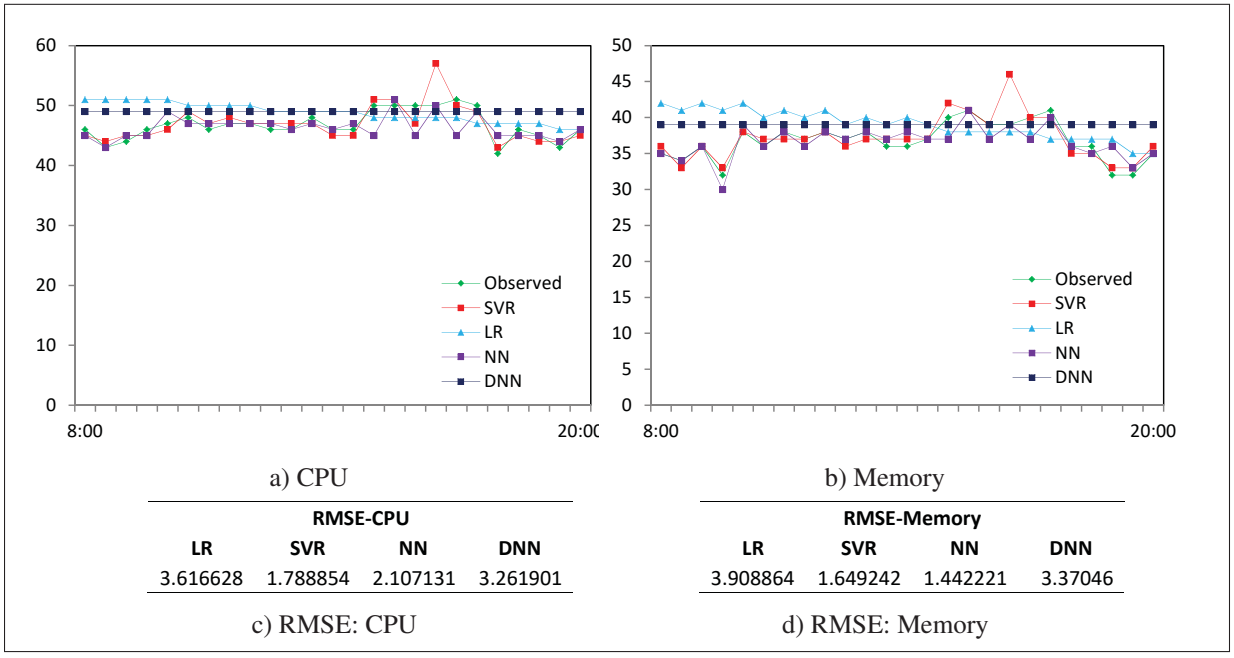
RMSE-CPU			
LR	SVR	NN	DNN
2.3749	0	1.8868	2.3833

c) RMSE: CPU

RMSE-Memory			
LR	SVR	NN	DNN
2.4739	0	1.3416	2.5298

d) RMSE: Memory

Figure 5.6 DS3: observed vs. predicted values.



a) CPU

b) Memory

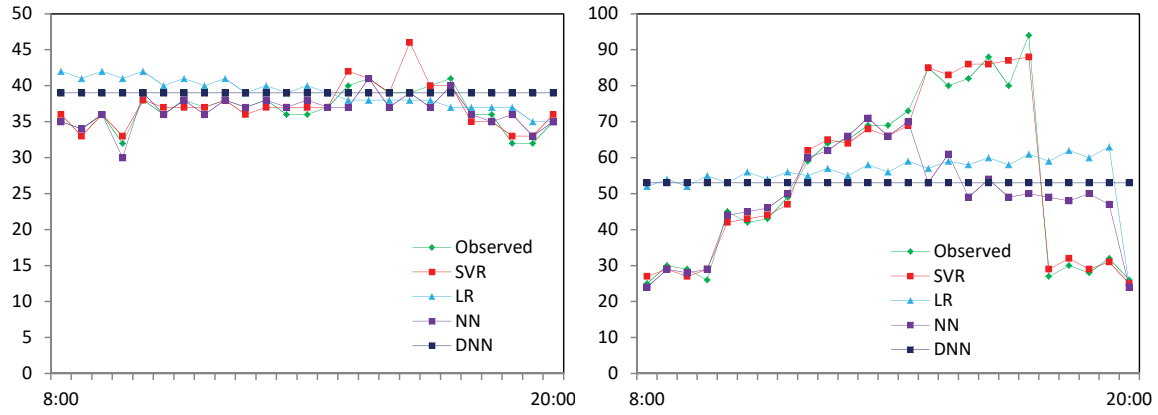
RMSE-CPU			
LR	SVR	NN	DNN
3.616628	1.788854	2.107131	3.261901

c) RMSE: CPU

RMSE-Memory			
LR	SVR	NN	DNN
3.908864	1.649242	1.442221	3.37046

d) RMSE: Memory

Figure 5.7 DS4:Observed vs. Predicted values.



a) CPU

b) Memory

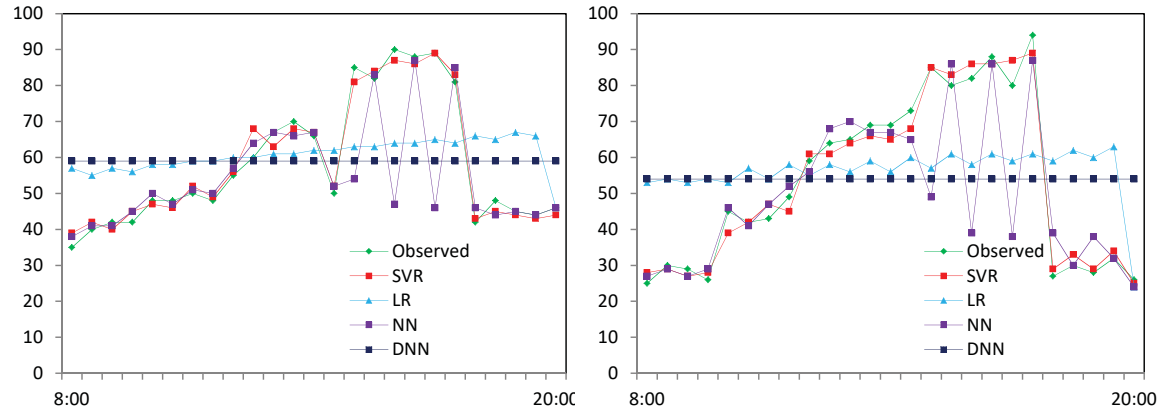
RMSE-CPU			
LR	SVR	NN	DNN
16.28619	2.481935	17.78089	17.5294

c) RMSE: CPU

RMSE-Memory			
LR	SVR	NN	DNN
21.86413	2.792848	18.00889	23.11796

d) RMSE: Memory

Figure 5.8 DS5:Observed vs. Predicted values.



a) CPU

b) Memory

RMSE-CPU			
LR	SVR	NN	DNN
16.28373	2.727636	13.82751	17.5294

c) RMSE: CPU

RMSE-Memory			
LR	SVR	NN	DNN
21.72464	3.298485	14.71462	23.11363

d) RMSE: Memory

Figure 5.9 DS6:Observed vs. Predicted values.

The next experiments are performed based on the values predicted by SVR since the latter had the highest accuracy in all previous conducted analysis. In what follows, we study the efficiency of the proposed optimization and dynamic programming algorithm to find efficient management strategies in different scenarios that reflect various usage pattern of the mobile device.

We compare the results with previous existing work, which adopts heuristics to find the strategy for each component. Table 5.6 illustrates the configuration of each scenario.

Table 5.6 Usage scenarios.

	Scenario-1	Scenario-1	Scenario-3	Scenario-4
Total VPs Running	1	3	2	3
Total Components Running	2	10	8	10
Predicted CPU Needs	10%	80%	60%	30%
Predicted Memory Needs	20%	70%	70%	30%

Table 5.7 shows the strategies generated for each scenario, by both heuristic and dynamic programming decision engines. To recall, a 0 bit in the heuristic approach is for local execution of the component while a bit of 1 is for offloading it. In the dynamic programming decision engine, the '||' is to separate strategies between VPs and '|' to separate strategies of components in the same VP.

Table 5.7 Generated strategies.

Scenario	Heuristic	DPDE
Scenario-1	1 1	[1,1,1] | [1,1,1]
Scenario-2	0 1 0 0 1 1 1 1 0 0	[1,0,-] || [1,1,1] | [1,1,1] | [1,1,1] | [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-]
Scenario-3	0 0 1 0 0 1 0 0	[1,1,0] | [1,1,0] | [1,0,-] || [1,1,0] | [1,0,-] | [1,1,0] | [1,0,-] | [1,0,-]
Scenario-4	0 0 0 0 1 1 1 1 1 1	[1,0,-] || [0,0,-] | [0,0,-] | [0,0,-] || [0,0,-] | [0,0,-] || [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-]

The resource consumption and performance of each of the applied strategies are compared in Figure 5.10.

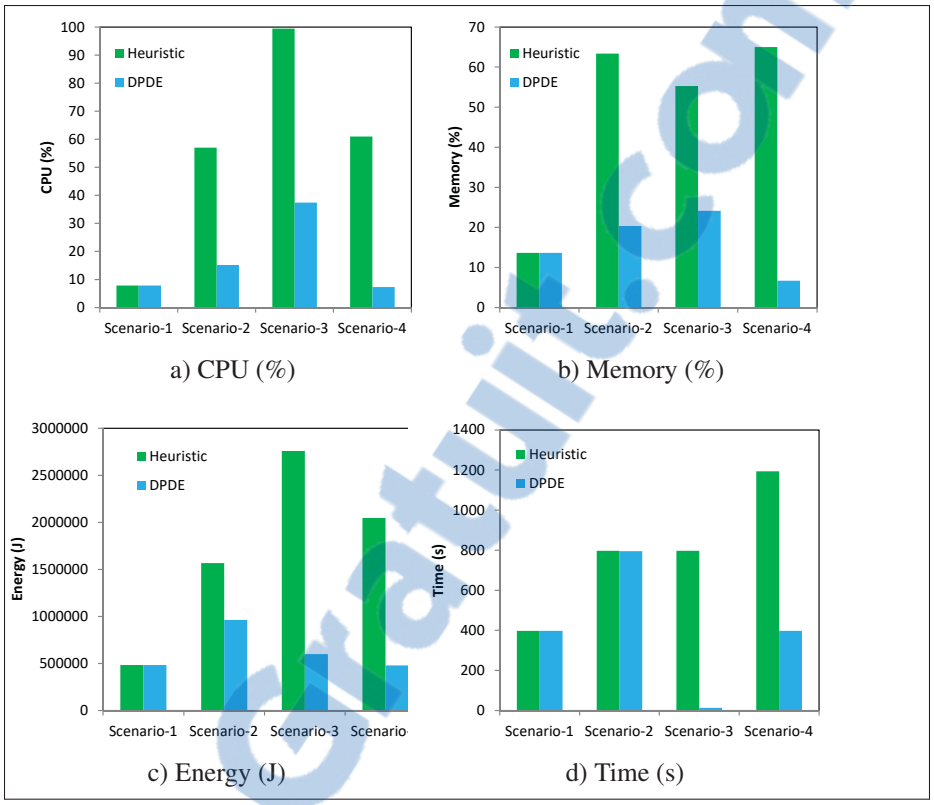


Figure 5.10 Strategies efficiency.

The figure shows that the algorithm proposed in this work is able to find better strategies with less CPU, memory and energy consumption and better performance compared to the heuristic approach. But how efficient are these strategies with respect to the predicted resource needs in each scenario? To answer this question, we examine Figures 5.10a and 5.10b. For Scenario-1, the results show that both approaches, i.e., Heuristic and DPDE are able to find a strategy that meets with both CPU and memory requirements. Particularly, The strategies found by both approaches imply 7.83% CPU and 13.634% memory which guarantees the availability of the needed 10% and 20% of CPU and memory respectively. For Scenario-2, 80% available CPU and 70% available memory are required as depicted in Table 5.6. The strategy found by the heuristic approach consumes 56.96% CPU and 63.39% memory which does not meet with the future resource needs. With DPDE, the strategy found only consumes 15.17% of the CPU and 20.34% of the memory, which keeps more available resources that meet with the resource required in future context. The same applies on Scenario-3. Finally for Scenario-4, the results

show that both approaches are able to find good strategies that meet with the device needs yet with better results of the management strategy generated by DPDE, which guarantees more resource availabilities of 92.67% CPU and 93.3% of the memory usage. These results prove the efficiency of the advanced management strategies proposed in this work to manage predicted future resource needs.

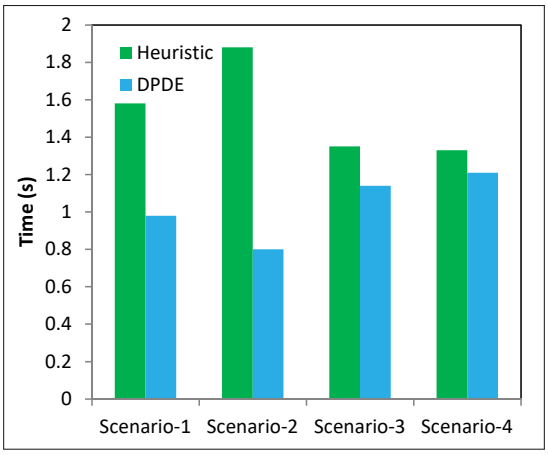


Figure 5.11 Decision engine performance.

Finally, we study the performance of heuristic and DPDE algorithms in terms of execution time. Figure 5.11 shows that for all sceanrios in question, DPDE shows faster execution.

5.9 Conclusion and Future Directions

Managing virtual phones (VPs) running on an end physical mobile device with limited resources is challenging. In this context, we proposed in this work novel approach able to predict future context and resource needs of these VPs and apply advanced management strategies accordingly, aiming to avoid any performance degradation or system crash in the system. Various machine learning techniques are studied and the one with the highest accuracy is adopted. Additionally, new management strategies are proposed and novel algorithm based on dynamic programming is presented to generate the adequate strategies that meet with the resource needs according to different usage scenarios. Thorough analysis was conducted to study the effi-

ciency of this proposition. The results proved the efficiency of the predictor, the adequacy of the new strategies proposed and the competency of the algorithm performance. As for the research community, studying the effect and overhead of training the model and performing the prediction on the end terminal would be interesting. Also, examining the effect of the window size value on the accuracy of the prediction model and proposing a dynamic generic approach to adapt this value to different data sets would be a valuable future research track.

Acknowledgment

The work has been supported by École de Technologie Supérieure (ETS), NSERC Canada, the Associated Research Unit of the National Council for Scientific Research CNRS Lebanon and the Lebanese American University (LAU).

CONCLUSION AND RECOMMENDATIONS

This section concludes this doctoral research work. It summarizes the main addressed problems and the presented key contributions, while opens the door for different future research directions.

While multi-persona has become an essential utility and virtualization has been ported on mobile terminals, co-hosting multiple virtual phones on a single mobile device is still impeded by the resource constraints of the latter hardware, which has limited computation capabilities, memory capacity and battery lifetime, compared to its counterpart of desktop machines. Addressing this problem has drawn the fundamental objective of this thesis. These impediments and their implications of putting personas performance and viability on the line have been studied and addressed throughout different models, techniques and algorithms proposed in each chapter.

Throughout the work, various research questions has arisen and many sub-objectives had to be set in order to attain the main goal. After studying existing virtualization techniques, extensive experiments have been performed proving the limitations of the mobile terminal resources to support multi-persona without causing performance degradation or shorter-time system viability. Afterwards, latest technologies that came out to support mobile devices, more specifically, mobile cloud computing, were investigated. Based on the premises of the benefits concluded in this area, we started to build our solution.

In the first article (Chapter 2), we proposed an offloading-based architecture and studied its effectiveness in augmenting multi-persona performance and viability. In this architecture, we were able to build generic optimization model, independent of the offloading granularity and adaptable to different execution contexts. We also presented a heuristic-based algorithm to automatically generate exact dissemination strategies for local and remote processing of multi-persona applications to optimize both resource usage and performance on the end terminal.

In the second article (Chapter 3), we proposed a selective solution for lightweight evaluation of computation offloading cost. Particulary, we proposed a novel selective mechanism based on hotspots to minimize the search space of potential components distribution strategies.

In the third article (Chapter 4), we proposed an approach to balance in one hand, the usage of cloud-based offloading services to minimize processing, memory, energy and execution time in personas on as many devices engaged in an organization as possible, and on the other hand the remote execution fees imposed on the institution itself when adopting such solution. For that end, we formulated two-level cost-effective optimization model and evaluated both centralized and decentralized decision making approaches to examine their engagement to address the raised concerns from different perspectives.

Finally, in the fourth article (Chapter 5), we highlighted the need for proactive approach able to assure future resource needs of the virtual instances and avoid system crash in the physical device. We advanced in this chapter a proactive scheme to predict resource needs in future context. We also proposed advanced strategies, beside offloading, which we emphasized on its limitations to solely manage VPs through different usage scenarios. Moreover, we presented a novel optimization model to meet with the resource needs, enhance the performance on the end device and support the proposed strategies. Further, a dynamic programming-based algorithm was introduced to find the adequate strategies to be applied by the end terminal. This approach allows managing situations with awareness for proactive and instructive recommendations. Rather than dictating what components to offload, more valuable recommendations can be taken based on user preferences, resource availabilities and device state. Such decisions include turning off applications, switching-off personas due to resource scarcity and direct decisions to prioritize the mobile device survivability over independent applications performance.

In each of these chapters, thorough analysis has been conducted and the results have proved the efficiency and competency of each of the proposed architectures, techniques, models and algorithms.

To summarize, this thesis offers the following contributions:

Technical Advancements:

- Mobile cloud-based architecture for efficient multi-persona mobile computing support.
- Cost-effective solution for multi-persona mobile computing in workplace.
- Generic, adaptable and lightweight optimization techniques for virtual phones' resource and performance management.
- Proactive method with advanced manageability strategies for efficient control of virtual phones' performance and viability.
- Competitive algorithms that automatically generate the adequate strategies to be applied by the end terminal.

Publications:

- Tout, H., Talhi, C., Kara, N. & Mourad, A. (2015). Towards an offloading approach that augments multi-persona performance and viability. *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pp. 455–460.
- Tout, H., Talhi, C., Kara, N. & Mourad, A. (2016). Selective Mobile Cloud Offloading to Augment Multi-Persona Performance and Viability.*Cloud Computing, IEEE Transactions on*.

- Tout, H., Talhi, C., Kara, N. & Mourad, A. (2017). Smart mobile computation offloading: Centralized selective and multi-objective approach. *Expert Systems with Applications*, 80, 1-13.
- Tout, H., Mourad, A., Kara, N. & Talhi, C. Cost-Effective Cloud-Based Solution for Multi-Persona Mobile Computing in Workplace (Journal Article Under Review)
- Tout, H., Kara, N., Talhi, C. & Mourad, A. Proactive Solution and Advanced Manageability of Multi-Persona Mobile Computing (Journal Article Under Review).

Each of the presented approaches opens the door for interesting **future tracks**:

- Monitoring the device state, applications running on it as well as the network conditions is essential to adopt the adequate management strategy on the device. Yet, investigating the frequency of calling these profilers would be intrusting to avoid any bottleneck they might cause in turn in the device resources.
- While assuming independent components on the device to reduce the complexity of the optimization model, considering the correlation between components of an application is important knowing that only few works have been proposed in this regard. Dynamic analysis of potential execution flow paths of different tasks in a mobile application has direct impact on the distribution decision where the decision to offload or locally execute particular components can influences the execution of other dependents components.
- While we adopted remote predictor, studying the effect and the overhead of centralizing the model training and prediction process on the end terminal would be interesting. Also, a fixed window size value based on extensive experiments was adopted to perform the prediction, yet proposing a dynamic automatic approach, generic enough to be adopted independently of the device usage data sets, would be a valuable future research track as well.

BIBLIOGRAPHY

Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A. & Buyya, R. (2014). Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1), 337–368.

Ahmed, E., Akhunzada, A., Whaiduzzaman, M., Gani, A., Ab Hamid, S. H. & Buyya, R. (2015a). Network-centric performance analysis of runtime application migration in mobile cloud computing. *Simulation Modelling Practice and Theory*, 50, 42–56.

Ahmed, E., Gani, A., Khan, M. K., Buyya, R. & Khan, S. U. (2015b). Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 52, 154–172.

Ahmed, E., Gani, A., Sookhak, M., Ab Hamid, S. H. & Xia, F. (2015c). Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, 52, 52–68.

Amazon. Amazon EC2 - Virtual Server Hosting. Consulted at https://aws.amazon.com/ec2/.

Amazon. Amazon EC2 Pricing. Consulted at https://aws.amazon.com/ec2/pricing/.

Andreev, S., Pyattaev, A., Johnsson, K., Galinina, O. & Koucheryavy, Y. (2014). Cellular traffic offloading onto network-assisted device-to-device connections. *IEEE Communications Magazine*, 52(4), 20–31.

Android. Android Interface Definition Language (AIDL). Consulted at http://developer.android.com/guide/components/aidl.html.

Android. (2014). The power of Android at work. Consulted at https://www.android.com/enterprise/employees/.

Android. (2016). The Activity Lifecycle. Consulted at https://developer.android.com/guide/components/activities/activity-lifecycle.html.

Android. (2017, March). Power Profiles for Android. Consulted at https://source.android.com/devices/tech/power/index.html.

Andrus, J., Dall, C., Hof, A. V., Laadan, O. & Nieh, J. Cells: Lightweight Virtual Smartphones. Consulted at http://systems.cs.columbia.edu/projects/cells/.

Andrus, J., Dall, C., Hof, A. V., Laadan, O. & Nieh, J. (2011). Cells: a virtual mobile smartphone architecture. *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 173–187.

Bala, A. & Chana, I. (2016). Prediction-based proactive load balancing approach through VM migration. *Engineering with Computers*, 32(4), 581–592.

Barr, K., Bungale, P., Deasy, S., Gyuris, V., Hung, P., Newell, C., Tuch, H. & Zoppis, B. (2010). The VMware mobile virtualization platform: is that a hypervisor in your pocket? *ACM SIGOPS Operating Systems Review*, 44(4), 124–135.

Beloglazov, A. & Buyya, R. (2010). Energy efficient allocation of virtual machines in cloud data centers. *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pp. 577–578.

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P. & Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Athena scientific Belmont, MA.

Cai, Z. & Chen, C. (2014). Demand-driven task scheduling using 2D chromosome genetic algorithm in mobile cloud. *Progress in Informatics and Computing (PIC), 2014 International Conference on*, pp. 539–545.

Cardellini, V., DE NITO, P. V., Di Valerio, V., Facchinei, F., Grassi, V., LO, P. F. & Piccialli, V. (2013). A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming*. Consulted at http://dx.doi.org/10.1007/s10107-015-0881-6.

Cellrox. (2013). Cellrox Partners with Fixmo to Bring Multi-Persona Solution for BYOD to iOS. Consulted at http://www.cellrox.com/press-release/cellrox-partners-with-fixmo-to-bring-multi-persona-solution-for-byod-to-ios.

Chae, D., Kim, J., Kim, J., Kim, J., Yang, S., Cho, Y., Kwon, Y. & Paek, Y. (2014). CM-cloud: Cloud platform for cost-effective offloading of mobile applications. *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pp. 434–444.

Chen, E., Ogata, S. & Horikawa, K. (2012). Offloading Android applications to the cloud without customizing Android. *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pp. 788–793.

Chen, M.-H., Liang, B. & Dong, M. (2015a). A semidefinite relaxation approach to mobile cloud offloading with computing access point. *Signal Processing Advances in Wireless Communications (SPAWC), 2015 IEEE 16th International Workshop on*, pp. 186–190.

Chen, W., Xu, L., Li, G. & Xiang, Y. (2015b). A lightweight virtualization solution for Android devices. *Computers, IEEE Transactions on*, 64(10), 2741–2751.

Chen, W., Xu, L., Li, G. & Xiang, Y. (2015c). A Lightweight Virtualization Solution for Android Devices. Consulted at http://dx.doi.org/10.1109/TC.2015.2389791.

Chen, X. (2015). Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(4), 974–983.

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

Cimino, M. G., Lazzerini, B., Marcelloni, F. & Ciaramella, A. (2012). An adaptive rule-based approach for managing situation-awareness. *Expert Systems with Applications*, 39(12), 10796–10811.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R. & Bahl, P. (2010). MAUI: making smartphones last longer with code offload. *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62.

Dall, C. & Nieh, J. (2013). *KVM/ARM: Experiences Building the Linux ARM Hypervisor* (Report n°CUCS-010-13). Consulted at http://academiccommons.columbia.edu/item/ac:162668.

Deb, K. (1999). An introduction to genetic algorithms. *Sadhana*, 24(4-5), 293–315.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2), 182–197.

Demuth, H. B., Beale, M. H., De Jess, O. & Hagan, M. T. (2014). *Neural network design*. Martin Hagan.

Deng, S., Huang, L., Taheri, J. & Zomaya, A. Y. (2015). Computation offloading for service workflow in mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(12), 3317–3329.

Dinh, H. T., Lee, C., Niyato, D. & Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18), 1587–1611.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J. & Vapnik, V. (1997). Support vector regression machines. *Advances in neural information processing systems*, pp. 155–161.

Eiferman, O. (2013). BYOD: What Containers and Wrappers Don't Tell You. Consulted at http://www.cellrox.com/blog/byod-what-containers-and-wrappers-dont-tell-you.

Eiferman, O. (2014a). MWC Blog Series Part III: Cellrox Drives Business by Embracing Changing Models of Employment. Consulted at http://www.cellrox.com/blog/mwc-blog-series-part-iii-cellrox-drives-business-embracing-changing-models-employment.

Eiferman, O. (2014b). How to balance security and freedom in medical BYOD. Consulted at http://health-information.advanceweb.com/Features/Articles/Physicians-Mobile-Devices.aspx.

Emmerich, M., Beume, N. & Naujoks, B. (2005). An EMO algorithm using the hypervolume measure as selection criterion. *Evolutionary Multi-Criterion Optimization*, pp. 62–76.

Fang, B., Liao, S., Xu, K., Cheng, H., Zhu, C. & Chen, H. (2012). A novel mobile recommender system for indoor shopping. *Expert Systems with Applications*, 39(15), 11992–12000.

Farahnakian, F., Liljeberg, P. & Plosila, J. (2013a). LiRCUP: Linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers. *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pp. 357–364.

Farahnakian, F., Pahikkala, T., Liljeberg, P. & Plosila, J. (2013b). Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data centers. *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, pp. 256–259.

Farahnakian, F., Ashraf, A., Pahikkala, T., Liljeberg, P., Plosila, J., Porres, I. & Tenhunen, H. (2015). Using ant colony system to consolidate VMs for green cloud computing. *IEEE Transactions on Services Computing*, 8(2), 187–198.

Fernando, N., Loke, S. W. & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future generation computer systems*, 29(1), 84–106.

Fiandrino, C., Kliazovich, D., Bouvry, P. & Zomaya, A. Y. (2015). Network-assisted offloading for mobile cloud applications. *Communications (ICC), 2015 IEEE International Conference on*, pp. 5833–5838.

Flores, H., Srirama, S. N. & Buyya, R. (2014). Computational offloading or data binding? bridging the cloud infrastructure to the proximity of the mobile user. *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pp. 10–18.

Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S. & Buyya, R. (2015). Mobile code offloading: from concept to practice and beyond. *Communications Magazine, IEEE*, 53(3), 80–88.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press.

Google. Google Cloud Platform. Consulted at https://cloud.google.com/.

Google. Google Compute Engine Pricing. Consulted at https://cloud.google.com/compute/pricing#custommachinetypepricing.

Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M. & Chen, X. (2012). COMET: code offload by migrating execution transparently. *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pp. 93–106.

Grefenstette, J. J. (2013). *Genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms*. Psychology Press.

Han, B., Hui, P., Kumar, V., Marathe, M. V., Pei, G. & Srinivasan, A. (2010). Cellular traffic offloading through opportunistic communications: a case study. *Proceedings of the 5th ACM workshop on Challenged networks*, pp. 31–38.

Huang, D., Wang, P. & Niyato, D. (2012). A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications*, 11(6), 1991–1995.

Hung, S.-H., Shieh, J.-P. & Lee, C.-P. (2012). Virtualizing Smartphone Applications to the Cloud. *Computing and Informatics*, 30(6), 1083–1097.

Hwang, J.-Y., Suh, S.-B., Heo, S.-K., Park, C.-J., Ryu, J.-M., Park, S.-Y. & Kim, C.-R. (2008). Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pp. 257–261.

Inc., E. BYOD: What Containers and Wrappers Don't Tell You. Consulted at http://www.enterproid.com.

Kemp, R. (2014). *Programming Frameworks for Distributed Smartphone Computing*. (Ph.D. thesis, VRIJE UNIVERSITEIT).

Kerrisk, M. (2013). Namespaces in operation, part 1: namespaces overview. Consulted at http://lwn.net/Articles/531114/.

Khan, A. R., Othman, M., Madani, S. A. & Khan, S. U. (2014). A Survey of Mobile Cloud Computing Application Models. *IEEE Communications Surveys Tutorials*, 16(1), 393-413. doi: 10.1109/SURV.2013.062613.00160.

Kodratoff, Y. (2014). *Introduction to machine learning*. Morgan Kaufmann.

Kosta, S., Aucinas, A., Hui, P., Mortier, R. & Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. *INFOCOM, 2012 Proceedings IEEE*, pp. 945–953.

Kovachev, D., Yu, T. & Klamma, R. (2012). Adaptive computation offloading from mobile devices into the cloud. *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pp. 784–791.

Liu, J., Kumar, K. & Lu, Y.-H. (2010). Tradeoff between energy savings and privacy protection in computation offloading. *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pp. 213–218.

Liu, Y. & Lee, M. J. (2014). An effective dynamic programming offloading algorithm in mobile cloud computing system. *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, pp. 1868–1873.

Lust, T. & Teghem, J. (2012). The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 19(4), 495–520.

Mahmoodi, S. E., Uma, R. & Subbalakshmi, K. Optimal Joint Scheduling and Cloud Offloading for Mobile Applications.

Mazza, D., Tarchi, D. & Corazza, G. E. (2014). A partial offloading technique for wireless mobile cloud computing in smart cities. *Networks and Communications (EuCNC), 2014 European Conference on*, pp. 1–5.

Menage, P. (2014). Cgroups. Consulted at https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.

NADLER, D. (2014). Multi-Persona Delivers on Changing Models of Practicing Medicine. Consulted at http://www.cellrox.com/blog/multi-persona-delivers-changing-models-practicing-medicine.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B. & Alba, E. (2009). Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7), 726–746.

Rouse, M. (2012a). BYOD (bring your own device). Consulted at http://whatis.techtarget.com/definition/BYOD-bring-your-own-device.

Rouse, M. (2012b). dual persona (mobile device management). Consulted at http://searchconsumerization.techtarget.com/definition/Dual-persona.

Rouse, M. (2014). COPE (corporate-owned, personally-enabled). Consulted at http://searchconsumerization.techtarget.com/definition/COPE-corporate-owned-personally-enabled.

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

Seber, G. A. & Lee, A. J. (2012). *Linear regression analysis*. John Wiley & Sons.

Shahzad, H. & Szymanski, T. H. (2016). A dynamic programming offloading algorithm for mobile cloud computing. *Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on*, pp. 1–5.

Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M. & Zegura, E. (2014). Cosmos: computation offloading as a service for mobile devices. *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pp. 287–296.

Shiraz, M. & Gani, A. (2014). A lightweight active service migration framework for computational offloading in mobile cloud computing. *The Journal of Supercomputing*, 68(2), 978–995.

Shiraz, M., Gani, A., Khokhar, R. H. & Buyya, R. (2013). A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(3), 1294–1313.

Sivanandam, S. & Deepa, S. (2007). *Introduction to genetic algorithms*. Springer Science & Business Media.

Song, J., Cui, Y., Li, M., Qiu, J. & Buyya, R. (2014). Energy-traffic tradeoff cooperative offloading for mobile cloud computing. *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*, pp. 284–289.

Spaces, S. Using Secure Spaces. Consulted at http://securespaces.com/.

Thede, S. M. (2004). An introduction to genetic algorithms. *Journal of Computing Sciences in Colleges*, 20(1), 115–123.

Toma, A. & Chen, J.-J. (2013). Computation offloading for frame-based real-time tasks with resource reservation servers. *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pp. 103–112.

Tout, H., Talhi, C., Kara, N. & Mourad, A. (2015). Towards an offloading approach that augments multi-persona performance and viability. *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pp. 455–460.

Tout, H., Talhi, C., Kara, N. & Mourad, A. (2016). Selective Mobile Cloud Offloading to Augment Multi-Persona Performance and Viability. *Cloud Computing, IEEE Transactions on*.

Tout, H., Talhi, C., Kara, N. & Mourad, A. (2017). Smart mobile computation offloading: Centralized selective and multi-objective approach. *Expert Systems with Applications*, 80, 1–13.

Wessel, S., Stumpf, F., Herdt, I. & Eckert, C. (2013). Improving Mobile Device Security with Operating System-Level Virtualization. In *Security and Privacy Protection in Information Processing Systems* (pp. 148–161). Springer.

Wu, C.-W., Chiang, T.-C. & Fu, L.-C. (2014). An ant colony optimization algorithm for multiobjective clustering in mobile ad hoc networks. *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pp. 2963–2968.

Wu, H. & Wolter, K. (2014). Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric. *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, pp. 90–97.

Wu, H., Wang, Q. & Wolter, K. (2013). Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. *Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 728–732.

Xia, F., Ding, F., Li, J., Kong, X., Yang, L. T. & Ma, J. (2014). Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16(1), 95–111.

Xiang, L., Ye, S., Feng, Y., Li, B. & Li, B. (2014). Ready, set, go: Coalesced offloading from mobile devices to the cloud. *INFOCOM, 2014 Proceedings IEEE*, pp. 2373–2381.

Xiao, Z., Song, W. & Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE transactions on parallel and distributed systems*, 24(6), 1107–1117.

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

Zhang, Y., Niyato, D. & Wang, P. (2015). Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, 14(12), 2516–2529.

Zhou, B., Dastjerdi, A. V., Calheiros, R., Srirama, S. & Buyya, R. (2016). mCloud: A Context-aware Offloading Framework for Heterogeneous Mobile Cloud. *IEEE Transactions on Services Computing*, (99). doi: 10.1109/TSC.2015.2511002.

Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N. & Buyya, R. (2015). A context sensitive offloading scheme for mobile cloud computing service. *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pp. 869–876.

Zitzler, E., Laumanns, M. & Thiele, L. (2002). SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, pp. 95–100.

Zitzler, E. & Künzli, S. (2004). Indicator-based selection in multiobjective search. *Parallel Problem Solving from Nature-PPSN VIII*, pp. 832–842.