Pós-Graduação em Ciência da Computação

“Improving Mobile Cloud Performance using Offloading Techniques and Stochastic Models”

By

Francisco Airton Pereira da Silva

Ph.D. Thesis

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEB/2017

Federal University of Pernambuco
Informatics Center
Pos Graduation in Computer Science

Francisco Airton Pereira da Silva

"Improving Mobile Cloud Performance using Offloading Techniques and Stochastic Models"

A Ph.D. Thesis presented to the Informatics Center of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Advisor: *Dr. Paulo Romero Martins Maciel*

RECIFE, FEB/2017

Acknowledgements

This thesis is the result of a four year journey, with 12 months of internship in Italy. I found, both at home and abroad, many new colleagues and friends. I would like to thank them all for their influence on the outcome of my PhD work, and ultimately on this thesis. I will mention some names briefly, as space dictates, and, if your name has been left out, please accept my non-nominative thanks.

I would like to start by thanking the Federal University of Pernambuco for showing me, as a researcher, that does not matter where we are, but who we want to become. As a substitute professor, the UFPE has shown me the importance of trying to achieve the highest student results.

I would like express my gratitude to professor Paulo Maciel, my adviser throughout the four years of the PhD. Paulo taught me how to do research always targeting the perfection. He gave me all support not only as an adviser but as a friend. Thank you for accepting me in your MoDCS research group and as collaborator at the EMC project. The MoDCS group represented an essential collaboration environment to reach my goals. I would like to mention my co-authors — Rubens, Jamilson, Bruno, Danilo, Iure and Thiago — for our mutual support and our continuous collaboration. I want to thank my colleagues — Maria Clara, Rosangela, Erico, André, Vandi, Ermeson, Gustavo, Aleciano, Verônica, Jean, Eliomar and João — for providing a stimulating research environment. A special thank for my Scientific Initiation undergraduate students — Gileno, Éder, Matheus and Germano — for helping me with the hard work.

My internship at the Sapienza University of Rome was an incredible experience. I had the opportunity of collaborating with highly professional researchers. I would like to thank professor Alessandro Mei for accepting me as a research visitor. My special thanks go to Sokol Kosta, for personally getting involved with my research and improve my results. Thanks also for my colleagues from the Lab — Irene, Miguel and Enis.

Last, but certainly not least, I would like to thank my friends from home, for their refreshing support, Renê, Jamilson, João, Alex, Hilário and Adriano. Foremost, I would like to thank mom, dad and the rest of my family, for being close to me even when I was far away and for their continuous support. I love you all very much!

When you walk through a storm,
Hold your head up high,
And don't be afraid of the dark.
At the end of a storm,
There's a golden sky,
And a sweet silver song of a lark
Walk On! Walk On! With hope in your heart,
And you'll never walk alone....

—RICHARD RODGERS (You'll Never Walk Alone)

,

Resumo

A escassez de recursos é um grande obstáculo para muitas aplicações móveis, uma vez que os dispositivos têm bateria e poder de processamento limitados. O uso da computação em nuvem tem se mostrado uma alternativa viável para processar cargas de trabalho de dispositivos móveis limitados. Com o objetivo de mitigar este problema nasceu o campo de pesquisa chamado computação em nuvem móvel (MCC). Ao usar a nuvem, os dispositivos móveis podem transferir seu processamento para servidores potentes. Muitas questões relacionadas a esse processo têm sido investigadas na última década, mas as relacionadas com o processo de execução remota ainda permanecem. Esta pesquisa de doutorado desenvolveu uma abordagem de execução remota de aplicativos móveis na nuvem. O algoritmo desenvolvido considerou uma estratégia inovadora de balanceamento de parâmetros coletados do estado da infraestrutura. Outro desafio do MCC está relacionado ao processo de avaliação e planejamento da infraestrutura tecnológica adotada. Uma avaliação detalhada do desempenho de diferentes configurações de infraestrutura pode fornecer aos engenheiros de software informações precisas, guiando suas decisões. Ao invés de avaliar a infraestrutura como uma caixa-preta, este trabalho propõe analisar a aplicação em nível de código-fonte, mais precisamente chamadas de método. O trabalho utiliza redes de Petri estocásticas (SPNs) para representar e avaliar desempenho e gasto de bateria de dispositivos móveis. As SPNs neste trabalho permitem aos engenheiros de software entender suas aplicações através de um relatório estatístico. Estudos de caso mostraram que as técnicas propostas nesta pesquisa são úteis para orientar designers e administradores de sistemas de nuvem no processo de tomada de decisão.

Palavras-chave: Computação em Nuvem Móvel, Redes de Petri Estocásticas, Offloading, Balanceamento de Carga, Avaliação de Performance, Energia

Abstract

Resource scarcity is a major obstacle for many mobile applications, since devices have limited battery and processing power. The use of cloud computing has been shown to be a feasible alternative to process demanding mobile devices workloads, leading to the research field called mobile cloud computing (MCC). By using the cloud, mobile devices may offload computation to resourceful servers. Many issues related to such a process have been investigated in the past decade, but those related to offloading process still remain. This PhD research has developed a smart MCC offloading strategy for mobile applications. The approach have considered an innovative balanced infrastructure parameters strategy. Another MCC challenge is related to the process of infrastructure evaluation and planning. Evaluating the MCC infrastructure in a deep level of detail may provide to software engineers precise information, guiding their decisions. Instead of evaluating the MCC infrastructure as a black-box, this work proposes to analyze the application at source-code level. This PhD research proposes providing a way for representing method-calls and evaluating mobile cloud applications by using stochastic petri nets (SPNs). The SPNs in this work allow software engineers to understand their applications through a statistic report. Case studies have showed that the proposed techniques are helpful for guiding cloud systems designers and administrators in the decision-making process.

Keywords: Mobile Cloud Computing, Stochastic Petri Nets, Offloading, Scheduling, Performance Evaluation, Energy

Contents

List of Figures

List of Tables

List of Acronyms

CDF Cumulative Distribution Function

CMD Cloudlet Manager Detection

CMTC Cloudlet Manager Threads Creation

CTMC Continuous Time Markov Chain

DoE Design of Experiment

EtpC Execution time per Core

MCC Mobile Cloud Computing

MTTE Mean Time to Execute

NF Number of Human Faces

PT Processing Time

QoS Quality of Services

RCM Return Result to Cloudlet Manager

RD Return Result to Device

RN Resource Pool Number

RTT Round-Trip Time

SCM Send Photo to Cloudlet Manager

SPN Stochastic Petri Net

SVM Send Pictures to the Virtual Machines

Td Transition Delay

Tp Throughput

TT Transmission Time

U CPU Utilization

VMR Virtual Machines Face Recognition

1 Introduction

Over the past few years, advances in the field of computer networks and operating systems virtualization led to an explosive growth of sophisticated architectures which started to provide services with high scalability and elasticity. This architecture, called Cloud Computing, has become an important area of scientific research and industry advancements since 2006, when the Amazon EC2 was launched (Antonio, 2013). Commonly, Cloud Computing is described as a range of services that are provided by a cluster system based on the Internet. Such cluster systems consist of a group of self-manageable servers that offer reliable, fast, convenient and transparent services, such as data storage and processing.

Meanwhile, mobile devices began to connect to the Internet due to the rapid growth of wireless network technology, among other factors. Today, mobility is a key feature in the new generation of Internet, which provides a set of custom services through numerous terminals. Cloud computing is marketed as a utility service (e.g: Amazon EC2), similar to common products such as water, gas, or electricity. Thus, the development of mobile access and the evolution of the cloud services enabled the creation of a new field of study called Mobile Cloud Computing (MCC). It is observed today that MCC contributes significantly to our daily life increasing the capabilities of mobile devices, but creating numerous challenges where the main one is to combine the two technologies.

The action of moving the processing from the mobile device to remote servers is called offloading and it aims to increase the capacity of mobile devices. Offloading can optimize energy usage and improve performance in mobile systems, however this usually depends on many parameters such as bandwidth and delay. Many algorithms have been proposed to analyze these parameters and decide when, how, and where to offload. Although mobile devices are growing in computing power, the role of more powerful infrastructures will increase, needing more sophisticated offloading techniques.

1.1 Context of Mobile Cloud Offloading

Modern handheld devices, such as smartphones and tablets, offer portability, increased computational power, and communication capabilities. These mobile devices are becoming an attractive option for users to interact with each other. On the other hand, along with the technological advances in hardware and mobile computing, user demands are also increasing, as they expect content rich applications, and access to large amounts of remote data, like multimedia streaming. Advanced as they may be, modern mobile devices still have some limitations in relation to user demands, in terms of battery supply, memory capacity and heat dissipation. Thus, it is reasonable to see why mobile devices, despite their increasing computing power, continue to use more powerful infrastructure.

The convergence of mobile and cloud computing has been studied for a number of years and is still a hot topic, because of the dynamics in mobile computing and because of the challenges that continue to arise (Araujo *et al.*, 2016; Matos *et al.*, 2015; Costa *et al.*, 2015; Abolfazli *et al.*, 2015; Chen *et al.*, 2015; Khan *et al.*, 2015; Lin *et al.*, 2015). As sales of mobile devices grow above sales of personal computers, many hardware and software manufacturers compete on the mobile market. Companies such as Samsung, Nokia, HTC, Motorola, Apple, Acer and Asus produce mobile devices of various hardware characteristics, using a variety of operating systems, either developed in house, like Apple's iOS, or by large software companies, like Google's Android or Microsoft's Windows. The heterogeneity of mobile devices, in terms of hardware and operating systems, makes it difficult for application developers to reach all the mobile users, and they usually have to maintain several versions of the same application. Cloud providers are also interested in tuning their systems to face big variations in the number of users. In this massive ecosystem, researchers find challenging topics with effects ranging from user experience to cost optimization for the resource providers.

Cloud offloading is one of the emerging trends in distributed computing involving mobile devices. Developers and researchers alike study ways of accessing, from user terminals, the power offered by cloud infrastructure in terms of storage. The cloud has been used for offloading storage and functionality for computers for a long time. However, more recently, mobile devices encouraged developments in computation and communication offloading, with a focus on the trade-offs between the benefits that the powerful infrastructure brings.

1.1.1 Offloading Concerns

Offloading aims to optimize the functionality of an application by using remote resources. Although most of the existing research efforts focus on key concerns, such as performance, energy and cost, some of them acknowledge that offloading is much more complex. Following a list of the main MCC concerns is discussed:

- **Execution time** In order to offer a minor execution time to its clients, the offloading system should be fine tuned to have its own performance at peak efficiency. Performance analysis is a complex task, but all of its aspects, ranging from modeling to measuring, have been researched and applied on various distributed systems.
- **Energy saving** is key for all modern computing systems. Mobiles are focused on energy saving due to their limited battery supply. Clouds are also focused on energy saving to ensure low costs for their users.
- **Costs**, from a financial point of view, can also become a complex aspect of offloading. A single offloading operation can imply costs for multiple service or resource providers, such as network operators, software manufacturers and cloud owners. The cost for data transmission can be extremely high.
- **Accuracy** of the results can also be a serious concern, especially when processing is done in parallel on the device and on a different architecture.
- **Heterogeneity** related to the significant number of mobile device brands and models. The development of technology that supports such variety is not straightforward.
- **Scalability** in offloading systems, as in any distributed system, is a serious concern when addressing large numbers of inputs. For example, if the offloading system goes public, it needs to scale well to ensure proper functionality for increasing numbers of users.
- **Elasticity** is the ability to adapt to workload changes and it usually involves actively creating and destroying resources. Thus, the system should either predict or react quickly to both positive and negative changes in the workload. If the system is not able to provision new resources, the clients will be affected by lack of service. If the system is not able to deprovision unused resources, then the financial cost will grow unnecessary for existing clients.

- **Customisation** refers to the property of a service to be customized to better serve the needs of various types of customers. Support for value adding operations like backup, update, cloning and avoid vendor lock-in.
- **Security** is, like in any distributed system, a topic of great interest, because data leaves the personal device and needs to travel over public infrastructure for remote processing.

1.1.2 Offloading Perspectives

Consider the scenario in which a mobile device does not support the execution of one specific application. To offload the workload to the cloud may be one option to make it feasible the execution whereas obeying user's constraints. MCC offloading systems usually explore one or more of the following three perspectives.

1. **What to Offload?** Considering that the application will be offloaded and the cloud supports many servers, then, to split and execute the application in parallel may be an option. However, the offloading system must decide at which level of granularity (class, method, components) should an application be partitioned.
2. **When to Offload?** The offloading system should decide whether it is worthy or not to execute the processes remotely. It does not make sense, for example, to offload one application through a very low quality Internet connection or when the application does not need much resources. Many parameters (such as connectivity and mobile device capability) should be considered when offloading, otherwise the application may waste performance.
3. **Where to Offload?** Usually the application is partitioned in a set of sub-parts. Next step is to decide for which machine these partitions should be sent. One data center has different types of servers in terms of resource power and technology. The characteristics of such a infrastructure should be considered to construct an offloading solution.

1.2 Research Scope and Motivation

Intending to provide meaningful findings, this PhD research focus on a subset problem associating specific concerns with perspectives in MCC offloading. The CCS Insight Institute forecasts that the global mobile phone market is expected to reach 2.35 billion units until 2019 (CCS-Insight-Forecast, 2015). This huge market-share stimulates mobile cloud research innovation aiming to satisfy more and more demanding users. Today, many applications that benefit from using the cloud have real-time constraints. These constraints become hard to meet expectations, mainly considering sophisticated cloud infrastructures.

The cloud may encompass heterogeneous components, utilizing virtual and physical machines with diverse computation power. High number of resources also can be challenging to manage, whereas powerful data centers become affordable for even small companies building large infrastructures. These data centers, when providing services to mobile devices, may tackle communication issues. Resources can be, for example, geographically distributed, and factors such as latency and intermittent connectivity must be considered by offloading systems. All the aspects discussed above are directly related to the “Where offloading perspective". As illustrated in Figure 1.1, this PhD research have focused mainly on the “where" perspective due to its challenging and important features.

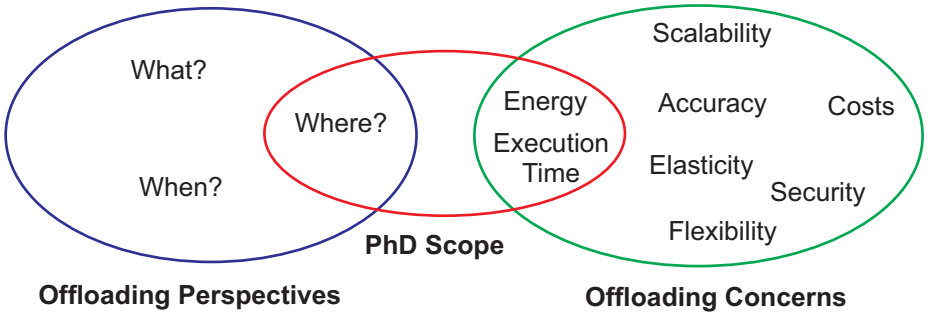


Figure 1.1: PhD Research Scope.

In terms of offloading concerns, this PhD research have focused on execution time and energy saving. They have always been users’ requirements and consequently a mobile industry interest. Offloading becomes an attractive solution for meeting response time requirements on mobile systems as applications become increasingly complex (Balan, 2006). A navigating robot application, for example, needs to recognize an object before it collides with the object; if the robot’s processor is too slow, the computation may

need to be offloaded (Nimmagadda *et al.*, 2010; Se *et al.*, 2005). Another application is context-aware computing (Hong and Landay, 2001) - where multiple streams of data from different sources like GPS, maps, accelerometers, temperature sensors, etc. need to be analyzed together in order to obtain real-time information about a user's context. In many of these scenarios, the limited computing speed of mobile systems can be enhanced by offloading Chun *et al.* (2011b).

In another hand, the advances in smartphone battery life have been slow to respond the computational demands of applications over the years. Many applications are still unsuitable for smartphones due to hardware constraints (Khan *et al.*, 2014). Computing speeds of these mobile devices, however, will not grow at the same pace as servers' performance. This is due to several constraints, including: Hardware constraints, as users want devices that are smaller and thinner and yet with more computational capability; Power consumption, insofar the current battery technology constrains the clock speed of processors, doubling the clock speed approximately octuples the power consumption. Consequently, it is difficult to offer long battery lifetimes with high clock speeds (Kumar *et al.*, 2013). Therefore, execution time and energy will continue being a MCC concern in long term, motivating further research under these topics.

1.3 Problem Statement

Consider the scenario illustrated in Figure 1.2. Imagine that a software engineer intends to build a MCC infrastructure. There is a mobile application that runs heavy tasks. Offloading these tasks may improve the application performance. One problem here is the distribution of tasks, aiming to use all the available resources. The current states of the target machines are diverse in terms of Round Trip Time (RTT) and current CPU consumption. In this context, two questions arise:

- How to distribute tasks considering multiple metrics (e.g.: RTT, CPU, etc.) in MCC?
- How to evaluate the performance and energy of MCC infrastructures?

1.4 Objectives

The main objective of this research is to develop new approaches in mobile cloud computing that can lead to performance and energy saving in mobile devices.

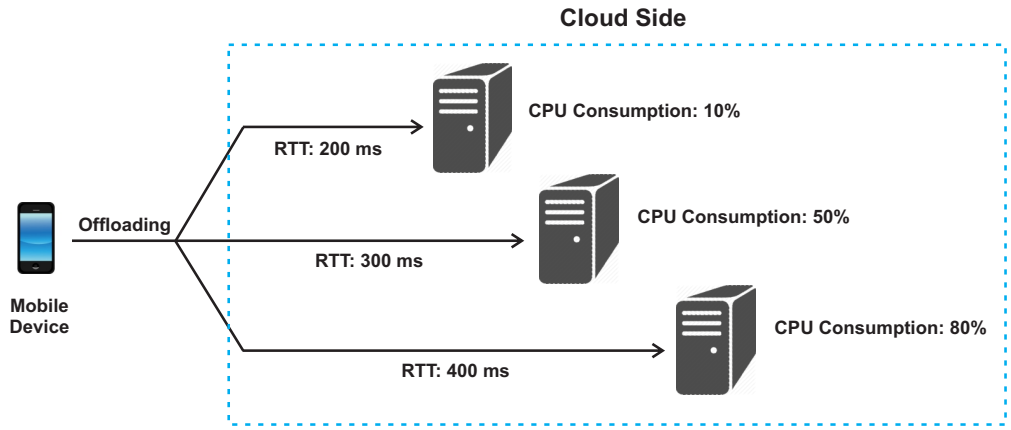


Figure 1.2: Problem Illustration.

Among the specific goals of the research, we can list:

1. Develop an algorithm for load balancing in MCC that can consider multiple metrics.
2. Develop an approach for evaluating MCC applications using SPNs.
3. Implement tools based on the proposed theories for assisting MCC application offloading.

1.5 Publications

Following, a list with the published papers related to this research is presented.

As main author:

- Francisco Airton Silva, Paulo Maciel, Eder Quesado, Rubens Matos, Jamilson Dantas. Mobile Cloud Face Recognition Based on Smart Cloud Ranking Journal of Computing , 2016.
- Francisco Airton Silva, Germano Zaicaner, Eder Quesado, Matheus Dornelas, Bruno Silva and Paulo Maciel Benchmark Applications Used in Mobile Cloud Computing Research: A Systematic Mapping Study The Journal of Supercomputing, 2016.
- Francisco Airton Silva, Paulo Maciel, Rubens Matos SmartRank: a smart scheduling tool for mobile cloud computing The Journal of Supercomputing, April, 2015.

- Francisco Airton Silva, Sokol Kosta, Matheus Rodrigues, Alessandro Mei, and Paulo Maciel. Planning Mobile Cloud Infrastructures Using Stochastic Petri Nets and Graphic Processing Units. In: Proceedings of 7th IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM). November 30 December 3, 2015.

- Francisco Airton Silva, Paulo Maciel, Eder Quesado, Germano Zaicaner, Matheus Dornelas, Bruno Silva Benchmark Applications Used in Mobile Cloud Computing: A Systematic Mapping Study The Twentieth IEEE Symposium on Computers and Communications (ISCC), 2015.

- Francisco Airton Silva, Paulo Maciel, Rubens Matos, Gileno Filho A Scheduler For Mobile Cloud Based on Weighted Metrics and Dynamic Context Evaluation 30th ACM/SIGAPP Symposium On Applied Computing (SAC), 2015.

As co-author:

- Eliomar Campos, Rubens Matos, Francisco Airton Silva, Francisco Vieira, and Paulo Maciel. Stochastic Modeling of Auto Scaling Mechanism in Private Clouds for Supporting Performance Tunning. In: IEEE Int. Conference on Systems, Man, and Cybernetics (IEEE SMC 2015). October 09-12, 2015, Hong Kong.

- Eliomar Campos, Rubens Matos, Paulo Maciel, Igor Costa, Francisco Airton Silva and Francisco Souza. Performance Evaluation of Virtual Machines Instantiation in a Private Cloud. In: Proceedings of IEEE 11th World Congress on Services (IEEE SERVICES 2015). June 27, July 02, 2015. New York, USA.

- Igor Costa, Jean Araujo, Jamilson Dantas, Eliomar Campos, Francisco Airton Silva, and Paulo Maciel. Availability Evaluation and Sensitivity Analysis of a Mobile Backend-as-a-Service Platform. Journal Quality and Reliability Engineering International. 2015. ISSN (online): 1099-1638.

- ARAUJO, C. ; SILVA, Francisco Airton Silva; COSTA, I. ; VAZ, F. ; KOSTA, S. ; MACIEL, P. R. M. . Supporting availability evaluation in MCC-based mHealth planning. Electronics Letters, p. 1-2, 2016.

1.6 Organization of the Document

This thesis is structured as follows. Chapter 2 clarifies some relevant background themes that the reader should know for properly understanding this document. Chapter 3 discusses noteworthy works found in literature that have some topics in common to those addressed in this thesis. Chapter **??** details the core contribution of this thesis. The Chapter describes an approach that uses stochastic models to evaluate mobile cloud performance and presents a mobile cloud offloading mechanism based on weighted metrics and dynamic context evaluation. Chapter 6 traces some conclusions and future work.

2 Background

This chapter discusses the basic concepts of mobile cloud and offloading mechanisms. The background presented here shall provide the necessary knowledge for a clear comprehension of the chapters ahead, including the aspects surrounding the proposed methodology and subsequent case studies.

2.1 Cloud Computing

Cloud computing is a paradigm in continuous development that originated from the combination of several different technologies. It has been defined as “a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers" (Mell and Grance, 2011; Buyya *et al.*, 2008).

A computational cloud is composed of five essential features (Abraham *et al.*, 2011):

- **On-demand self-service**: A consumer can obtain computing services (e.g.: server time and network storage) as needed, without requiring human interaction with each service provider;
- **Broad network access**: Capabilities are available over the network and accessed through standard mechanisms that promote the use by heterogeneous thin or thick client platforms;
- **Resource pooling**: The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual

resources dynamically being assigned and reassigned according to consumer demand.

- **Rapid elasticity**:Capabilities can be elastically provisioned and released, in some cases automatically, to rapidly scale outward and inward, and adjust the consumption of resources to the system's workload;
- **Measured service**: Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service.

The basic principle of cloud computing is to assign the computing to a large number of distributed computers, rather than local computers or remote services. It is characterized by the efficient utilization of resources, employing virtualization, resources monitoring, and load balancing mechanisms (Saranya and Vijayalakshmi, 2011).

On modern societies, the majority of essential services is made available on a transparent way. The water supply, electric power, gas and telephone, essential goods in our daily life, have this characteristic. These market models follow the concept of "pay for what you use": the paid value for the service is flexible in accordance to the necessity of the organization at any time (Gomes, 2012). Cloud computing provides a similar payment model for the utilization of computing services.

There are three models of implementation of Cloud Computing (Huang *et al.*, 2010). *Private cloud* is a cloud infrastructure provisioned for exclusive use by a single organization comprising multiple consumers. In the *public cloud* model, the cloud infrastructure is provisioned for open use by the general public that remains unique entities, but they are bound together by standardized or proprietary technology that enables data and application portability. *Hybrid cloud* model is the composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities. However, the hybrid clouds introduce additional complexity, the distribution of applications by both models (Subramanian, 2011).

Briefly, among the benefits associated to the utilization of the services on the cloud, we could highlight: the centralized management, the reduction of energetic consumption, and the decrease of inherent costs to the maintenance of traditional infrastructures. The cloud provides a diversity of services that favors the agility of market (Miller, 2008; Terry, 2011).

2.2 Mobile Cloud Computing

Similar to cloud computing, smartphones are also gaining enormous popularity due to the support for a wide range of applications, such as games, image processing, video processing, e-commerce, and online social network services (Kocjan and Saeed, 2012). The smartphone applications complexity grows in parallel with their demand on computing resources. The advances in smartphone hardware and battery life have been slow to respond to the computational demands of applications evolving over the years.

Many applications are still unsuitable for smartphones due to constraints, such as low processing power, limited memory, unpredictable network connectivity, and limited battery (Khan *et al.*, 2013). The combination of cloud computing, wireless communication, portable computing devices, location based services, mobile Web, etc., has laid the foundation for a novel computing model, called mobile cloud computing, which allows users an online access to unlimited computing power and storage space. Taking the cloud computing features in the mobile domain, (Kovachev *et al.*, 2011) defines “mobile cloud computing (MCC) as a model for transparent elastic augmentation of mobile device capabilities via ubiquitous wireless access to cloud storage and computing resources. MCC should provide dynamic adjusting of offloading in respect to change in operating conditions, while preserving available sensing and interactivity capabilities of mobile devices" .

Mobile cloud computing can be presented in many ways. In this research, we refer to MCC as the set of techniques that use cloud resources to empower mobile applications. Generally observing MCC resources it can be presented in two perspectives: (a) infrastructure based, and (b) ad-hoc mobile cloud (Khan *et al.*, 2013). In infrastructure based mobile cloud, the hardware infrastructure remains static and provides services to the mobile users. Alternatively, ad-hoc mobile cloud refers to a group of mobile devices that acts as a cloud and provides access to local or Internet based cloud services to other mobile devices. In this research, we limit the selection of application models to the former case namely, the infrastructure based mobile cloud. Therefore, ad-hoc mobile cloud based systems/application models and associated issues, such as mobility of cloud infrastructure and geo-distribution of service nodes (Huerta-Canepa and Lee, 2010), are beyond the scope of this work.

2.2.1 Mobile Cloud Offloading

Cloud offloading is one of the emerging trends in distributed computing involving mobile devices. Developers and researchers alike study ways of accessing, from user terminals, the power offered by cloud infrastructure in terms of storage. The cloud has been used for offloading storage and functionality for computers for a long time. However, more recently, mobile devices encouraged developments in computation and communication offloading, with a focus on the trade offs between the benefits that the powerful infrastructure brings and the costs, in time and money, of using remote resources.

Offloading has gained big interest in mobile cloud computing research, because it has similar aims as the emerging cloud computing paradigm, i.e. to overcome mobile devices shortcomings by augmenting their capabilities with external resources. Offloading or augmented execution refers to a technique used to overcome the limitations of mobile phones in terms of computation, memory and battery. Such applications, which can adaptability be divided in parts and offloaded are called elastic mobile applications (Kemp *et al.*, 2012), (Zhang *et al.*, 2010). Basically, this model of elastic mobile applications enables the developers the illusion as if he/she is programming virtually much more powerful mobile devices than the actual capacities. Moreover, elastic mobile application can run as standalone mobile application but also use external resources adaptively. Which portions of the application are executed remotely is decided at runtime based on resource availability. In contrast, client/server applications have static partitioning of code, data and business logic between the server and client, which is done in development phase (Kovachev *et al.*, 2011).

According to (Olteanu and Tapus, 2013), the offloading process usually is divided into three modules: decision, allocation and operation:

- **Decision** gathers some of the most diverse ideas in offloading for mobile devices, depending on the benefit assessment. The approaches differ in the way they assess benefits, how they collect feedback from previous iterations of the offloading process and how they take into account context.

- **Allocation** refers to the way in which the system decides on what resources to use for which tasks allocation criteria and how to use multiple tasks on a limited number of resources allocation strategy.

- **Operation** The offload operation itself can be met in a variety of conditions, depending on the theoretical mechanism used and the actual implementation. Of-

floading inherently implies a sort of division between what is done locally and what is done remotely. The division can refer either to data or processing tasks.

Offloading Benefits

Offloading has a number of benefits, some already exposed in commercial applications, other still only shown in research studies. Offloading addresses some of the limitations of mobile devices. For example, to prevent mobiles from performing numerous queries to services, the major mobile operating systems developers implemented push notification services, a form of communication offloading. Data and content offloading opened the way for feature recognition applications such as Shazam, that rely on massive amounts of data, that could not exist on a single device.

From the developer‘s perspective, besides increasing performance, offloading can ease the development process. The developer will not worry about the mobile device resource constrained and focus on implementation of core functionalities.

Offloading can be used for various purposes, to increase performance, to enable new functionality on mobile devices, or to enable new properties in mobile applications (like fairness in games). Offloading can also make it feasible to produce wearable devices, smaller, less capable devices, focused on a single function, like collecting statistics for joggers. Such devices can use mobiles as their offloading target.

Applications Partitioning

Several works have explored mobile cloud applications partitioning (Eom *et al.*, 2012; Kosta and Aucinas, 2012; Kemp *et al.*, 2012; Cuervo *et al.*, 2010). The remote execution of mobile applications, namely offloading, seeks to get the best performance of response time as well as saving energy. Considered a starting point in offloading process, smart partitioning may optimize jobs distribution in the cloud. Many factors can be taken into account in MCC applications partitioning. According to (Liu *et al.*, 2015), these factors are: partitioning granularity, partitioning objective, partitioning model, programming language support, presence of a profiler, allocation decision, analysis technique, and use of annotation.

Partition granularity, in particular, refers to the portion of the application which represents one atomic unit. One application can be offloaded without even any partition, in this case for example, the atomic unit is the application as a whole. Some of the possible granularity levels are:

- **No partitioning**: The entire application is offloaded.
- **Method-Call level partitioning**: Partitioning occurs at the method of application.
- **Object level partitioning**: The object of an application is partitioned to prepare for cyber foraging.
- **Thread level partitioning**: Partitioning occurs at the threads of an application.
- **Class level partitioning**: Application is partitioned into classes for offloading.
- **Task level partitioning**: Application is partitioned according to task.
- **Component level partitioning**: Partitioning a group of classes which may or may not be coupled for outsourcing to the remote server.
- **Bundle level partitioning**: Groups of Java class of applications are partitioned.
- **Allocation-site level partitioning**: Partitioning occurs on the level of allocation site where all the objects at this particular site will be seen as a single unit.
- **Hybrid level partitioning**: The results of partitioning consist of different granularity.

To choose a partitioning technique considering concurrently energy saving and performance gain is not straightforward. Although one technique can provide a higher granularity, the energy saving depends on some other aspects. The total amount of injected workload may influence the energy consumption, partition size and capacity of the environment (e.g., servers and network devices).

Considering *module level partitioning*, the application have a complete copy (a clone) at a remote server. The applications usually do not need any modification under the clone and the physical device can run identical binaries. However, one disadvantage arises when the application running on the clone needs to access the physical device hardware or there is a user interaction. It is possible to transfer input/output data between the device and clone environment over the network, but this may result in negative impact on response time and battery lifetime.

Abstract levels of granularity with larger pieces result in simple offloading mechanisms that require low monitoring communication overhead. However, abstract level of granularity results in increased data transmission overhead and therefore increases security threats for outsourcing components of the mobile application. For example, the

migration of an entire application is more vulnerable to network threats in comparison to the method outsourcing. Considering security aspects, spying finer level code is less meaningful to attackers, so, root method and input are preferable partitioning techniques. The above considerations lead us to concentrate on application level instead of cloning a complete device environment.

Classes and methods represent interesting options to be offloaded taking into account inherent units. However, the number of such units restricts the level of granularity. Another problem is related to coupling. Considering the object-oriented paradigm, classes tend to be referenced by at least one other class, making it hard to split the application. Hence, classes and methods should first be decoupled before offloading, but there are many non-trivial constraints to decouple them. Hence, we propose not to decouple methods by refactoring, but identify the heaviest method(s) and offload its inner method-calls if possible.

2.3 Performance Evaluation of Systems

System administrators need to provide the highest performance at the lowest cost. A performance evaluation is necessary when a system administrator wants to compare a number of alternative configuration scenarios to find the best one. It is also used to compare two similar systems and decide which one is better for a given task. Performance evaluation can also help to determine how well a system is performing certain tasks, and if some improvements are necessary. Generally, evaluating the performance of a system means to verify its behavior according to a defined set of metrics. The researcher must select appropriate evaluation techniques (e.g.: *analytical modeling*, *simulation* or *measurement*), perform a statistical analysis to identify possible bottlenecks and propose improvement solutions. This work has applied a parametric sensitivity analysis from the analytical modeling with SPN and CTMC models, and measurements based on the (Design of Experiment) DoE technique.

2.3.1 Measurement

DoE technique allows to obtaining a maximum of information about a system, regarding many factors, with a reasonable number of experiments and effort (Jain, 2008; Montgomery and Montgomery, 1984). A set of experiment executions planned through DoE can be analyzed to determine if the factors have significant effects, or if the differences in

the observed effects are due to variations caused by measurement errors and not controlled parameters (Guimarães *et al.*, 2013; Jain, 2008; Montgomery and Montgomery, 1984).

This study adopts the *General Full Factorial Design*, which uses all possible combinations of levels for all factors, i.e., there are no limits to the number of factors and the number of levels. This type of DoE allows every configuration to be examined, so we can find the effects of all factors and their interactions, which is an advantage; the disadvantage is that the cost of analysis can be very high if the number of factors and levels is too high, and also considering that each of these experiments may have to be repeated several times. It is possible to reduce the number experiments by reducing the number of factors, and/or the number of levels for each factor, or using *Fractional Factorial Design* instead (Jain, 2008).

2.3.2 Continuous Time Markov Chains

As shown in Figure 2.1, Markov chains can be represented as a directed graph with labeled transitions, indicating the probability or rate at which such transitions occur. In Markov chains, the states represent different conditions that the system may follow. The transitions between the states indicate the occurrence of events (Silva *et al.*, 2013) (e.g.: the arrival of tasks, or completion of service). In Figure 2.1, a new task arrives with rate λ, and a server completes the task with rate μ. For example, Figure 2.1 depicts a model for a system with two servers that process incoming jobs. If we observe the number of busy servers as a time function, we can consider it as a random variable or function $X(t)$. Each modification of X over (t) is called *state* $X_n(t)$. The set of all possible states is the *state space* of the model. Thus, it is possible to find the transition probabilities from a state to its successor $X_{n+1}(t)$. For this, it is necessary to specify the probability distribution function of $X_n(t)$. Such sequences or random functions of time are called stochastic processes. Stochastic processes are processes in which the random variable changes its state over time (Jain, 2008; Maciel and Kim, 2011; Kleinrock, 1975). They are usually adopted to characterize systems whose behavior is inherently probabilistic (Silva *et al.*, 2013).

Analytical modeling may consider a random variable or several sequences or families of random variables. With only one random variable it is simple to know what is the probability of its states over time (stationary) probability or at a specific time (transient) probability. Those probabilities are obtained by computing the *distribution function*. However, when we represent a number of phenomena in a system, i.e., several random variables, the calculation may be complex, because it requires computing the *joint distri-*

bution function. On the other hand, the calculation of probabilities for a random variable can be simplified when applied to an *exponential distribution function* or *geometric distribution function*. Markov chains is a state space model widely adopted to work with such distribution functions, and therefore simplify the analysis of systems modeled through many random variables.

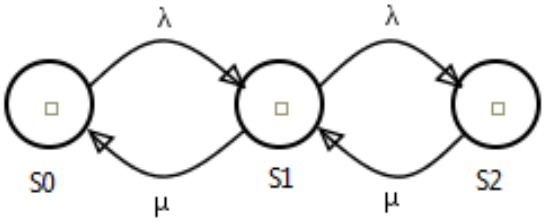


Figure 2.1: Example of a CTMC model

Markov chains are associated to a Markov process (Haverkort, 2002), and are stochastic models, used to analyze a variety of systems (Silva *et al.*, 2013). We have a Markov process if the past history is not important to know the probability of reaching a given future state. Only the current state is enough to know such a probability (property known as lack of memory). When the Markov process has a discrete state space, then it is known as a Markov chain. A Markov chain with discrete time parameter is called a DTMC. On the other hand, if the time parameter assumes real values, the model is called a CTMC (Jain, 2008; Maciel and Kim, 2011; Stewart, 1994). In a homogeneous DTMC, the time spent in a state follows a geometric distribution, while in the homogeneous CTMC follows an exponential distribution. Markov chains are said to be homogeneous, when the transition probability between states does not depend on time but only on the current state (Maciel and Kim, 2011). Markov chains have been used extensively in dependability, performance, and performability modeling (Maciel and Kim, 2011; Trivedi, 2001). CTMC was a useful modeling formalism for evaluating the performance of the cloud system studied in this work.

2.3.3 Stochastic Petri Nets

Petri nets (PNs) are a graphical and mathematical modeling tool applicable to many systems. They are promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, PNs can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. In addition,

tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Since Petri's seminal work, many representations and extensions have been proposed allowing more concise descriptions and representing systems features not observed on the early models (Murata, 1989).

SPNs are special cases of PNs. SPN models were proposed with the goal of developing a tool that allowed the integration of formal description, proof of correctness, and performance evaluation. The proposals regarding performance evaluation aimed at an equivalence between SPN and Continuous Time Markov Chains (CTMC) (German, 2000). In order to obtain an equivalence between a PN and a CTMC, it was necessary to introduce temporal specifications such that the future evolution of the model, given the present marking, is independent of the marking history. Therefore, SPNs can be translated to CTMC, which may then be solved to reach the desired performance or dependability results (Molloy, 1982; Marsan *et al.*, 1994; Trivedi, 2001; Marsan, 1990).

Figure 2.2 exhibits components used to model an SPN, and Figure 2.3 depicts an example of an SPN model. Places are represented by circles, whereas transitions are depicted as filled rectangles (immediate transitions) or hollow rectangles (timed transitions) or gray rectangles (unrefined transitions). The gray rectangle, in particular, has no associated time yet. It is used to represent that no experiment was executed to collect the time for that transition. Arcs (directed edges) connect places to transitions and vice versa. Tokens (small filled circles) may reside in places, which denote the state (i.e., marking) of an SPN. An inhibitor arc is a special arc that depicts a small white circle at one edge, instead of an arrow, and they usually are used to disable transitions if there are tokens present in a place. The behavior of an SPN is defined in terms of a token flow. Tokens are created and destroyed according to the transition firings (German, 2000). Immediate transitions represent instantaneous activities, and they have higher firing priority than timed transitions. Such transitions may also contain a guard condition, and a user may specify a different firing priority among other immediate transitions. There are also guard functions in SPNs. Guard functions are boolean expressions that control the firing of a transition, declaring some condition regarding the net's marking. If a transition's guard function produces a *true* value, it is able to fire, otherwise, the transition is disabled (Marsan *et al.*, 1994). Guard functions were not adopted in this work.

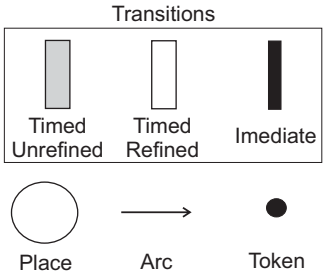


Figure 2.2: SPN Components.

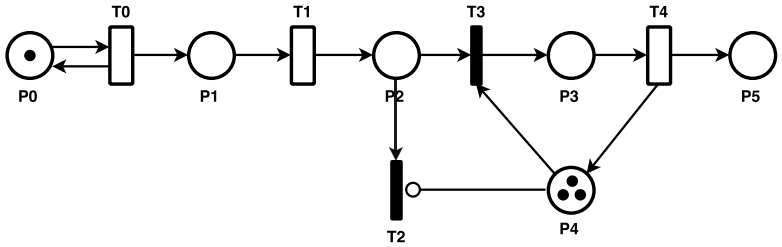


Figure 2.3: Example of an SPN model.

2.3.4 Phase-type approximation

Phase-type approximation methods (Desrochers *et al.*, 1995; Malhotra and Reibman, 1993) have been commonly used for representing the behavior of the unknown distribution functions, but it is also adopted for fitting distributions such as Erlang, hypoexponential and hyperexponential (Trivedi, 2001).

Measured data related to activities of systems (the respective average value μ_D and standard deviation σ_D, empirical distribution) may have their stochastic behavior represented by expolynomial distributions. Phase approximation technique may be applied through the inverse of coefficient of variation of measured data (Equation $\frac{1}{CV} = \frac{\mu_D}{\sigma_D}$) (Desrochers *et al.*, 1995). Analyzing the inverse of the coefficient of variation allows to choose the expolynomial distributions that best matches the measured data.

When the inverse of the coefficient of variation is a integer number and different from one, the empirical data should be characterized by an Erlang distribution that is represented in SPN model by a sequence of exponential transitions whose length is calculated by Equation $\gamma = \left(\frac{\mu_D}{\sigma_D}\right)^2$. The firing rate of each exponential transition is calculated by Equation $\lambda = \frac{\gamma}{\mu_D}$ (Desrochers *et al.*, 1995).

When the inverse of the coefficient of variation is a non-integer number larger than one, the empirical data is represented by a hypoexponential distribution which is however illustrated by a SPN model composed of a sequence whose length is calculated by

Equation $\left(\frac{\mu_D}{\sigma}\right)^2 - 1 \leq \gamma < \left(\frac{\mu_D}{\sigma}\right)^2$. Equations $\lambda_1 = \frac{\gamma}{\mu_1}$ and ($\lambda_2 = \frac{\gamma}{\mu_2}$ represent the firing rate of each exponential transition. The respective average delays (expected values) of the time assigned to the exponential transitions are calculated by Eqs. $\mu_1 = \mu_D \mp \frac{\sqrt{\gamma(\gamma+1)\sigma^2 - \gamma\mu^2}}{\gamma+1}$ and $\mu_2 = \gamma\mu_D \pm \frac{\sqrt{\gamma(\gamma+1)\sigma^2 - \gamma\mu^2}}{\gamma+1}$ (Desrochers *et al.*, 1995).

When the inverse of the coefficient of variation is a number smaller than one, the empirical data should be represented by an hyperexponential distribution. The firing rate of exponential transition should then be calculated by Equation $\lambda_h = \frac{2\mu_D}{{\mu_D}^2\sigma_D{}^2}$, in which the weights of immediate transitions are also calculated by Eqs. $\omega_1 = \frac{2{\mu_D}^2}{{\mu_D}^2\sigma_D{}^2}$ and $\omega_2 = 1 - \omega_1$ (Desrochers *et al.*, 1995).

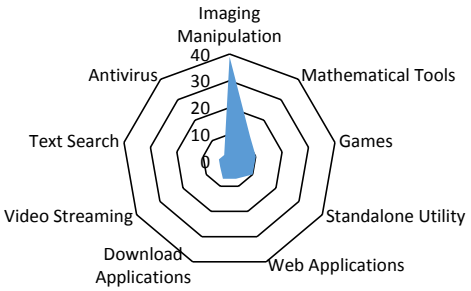
SPN models can be refined using phase-type approximation methods. In this work, hypoexponential distributions for phase approximation was adopted to obtain accurate SPN models and consequently computing mobile cloud performance metrics in a more precise manner.

2.4 Benchmark Applications used in MCC

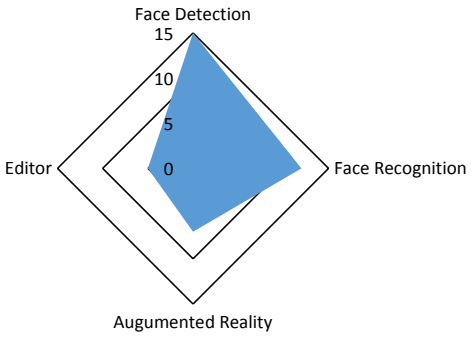
A significant amount of research has been performed on MCC offloading. Aiming to conduct these studies, most of the researchers have adopted real mobile applications to prove their hypothesis. However, there is no common list of which applications could be used in MCC research and a systematic mapping study could give important directions in this sense. A systematic mapping study is a type of investigation that has an evidence-based nature, applied in order to provide an overview of a research area by characterizing it (Petersen *et al.*, 2008). Before presenting any offloading strategy that could solve the pursued objective of this thesis we have applied a systematic mapping study, aiming to identify mobile applications used in MCC (Silva *et al.*, 2015a). We have executed a systematic mapping study by means of analyzing three applications' facets (Category, Platform, and Evaluated Resource). We synthesized implications for practicing, identifying research trends, open issues, and areas for improvement.

Starting from 763 papers, we filtered 47 studies that used applications as benchmarks. Given the current state of MCC research, we judge that there are few studies with controlled experiments using real applications. In our study, only 47 papers used applications to evaluate their proposals, probably because this field is still relatively recent, with the first effectively mobile cloud paper dating from 2009 (Liu *et al.*, 2009). In most of the cases, the studies did not provide evidences of how other researchers could access and download the applications used, making it hard to replicate their experiments. From the

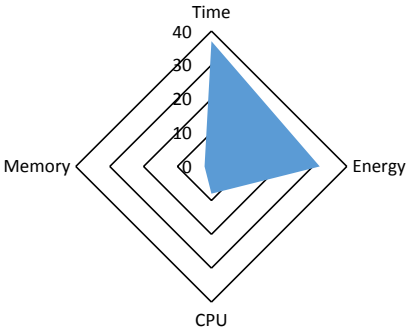
47 papers, we listed 25 downloadable applications with their corresponding category and URL. Following, by using radar plots, Figure 2.4 summarizes our findings. Radar plots display multivariate data in the form of a two-dimensional chart of n quantitative variables represented on axes, starting from the same point. The more distant from the center, more significant the result is — in this study, meaning higher quantity.



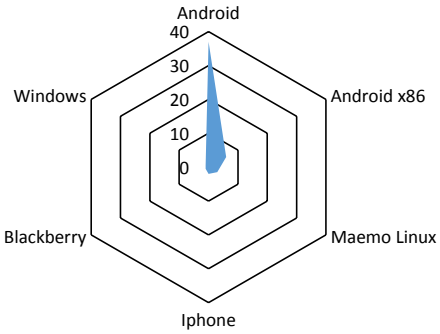
(a) Application Category.



(b) Imaging Manipulation.



(c) Evaluation Metric.



(d) Platform.

Figure 2.4: Mapping by Quantity of Occurrences.

Application category, illustrated in Figure 2.4a, means the functionality provided by mobile applications. Although games are well known as heavy processing applications, the imaging manipulation type was the most exploited in MCC so far. Due to the expressive quantity of papers using imaging manipulation category, we investigated more closely this topic in order to know what type of applications researchers have used more. Figure 2.4b shows that image detection and recognition were the most explored. Among the evaluated metrics (Figure 2.4c), as expected, energy and time were the most envisioned metrics, since among other motivations their impact is easily perceived by final users. Regarding platforms (Figure 2.4d), Android was the most employed platform in MCC, with 37 occurrences. All the decisions in this PhD research shall take into account these results, aiming to adopt similar test-beds and then make it possible to

compare results with related studies.

3

Related Work

The related work is presented in two sections, which references the two core contributions of this thesis: Evaluating/Planning MCC Applications and MCC Offloading. The following analysis does not intend to provide an exhaustive view of published works on those topics, but rather to point out significant advances which go towards a similar direction as this research do, or give basis for future extensions.

3.1 Evaluating and Planning MCC Applications

Table 3.1 synthesizes the contributions of the most prominent works related to this part of the thesis. The references are ordered by year (from 2009 to 2016) and encompass 17 studies, categorized by four aspects: Objective, Evaluated Metric, Modeling Granularity, and the use of Multiple Surrogates. For a better understanding we comment each aspect in details:

Objective - The first papers in MCC had the objective of optimizing the offloading process itself. They focused only on improving the offloading techniques by monitoring the mobile device, the application, and the network conditions. Many offloading frameworks have tackled mobile device constraints by offloading as much as possible heavy tasks obeying context factors (Kristensen, 2010; Cuervo *et al.*, 2010; Kemp *et al.*, 2012; Kosta and Aucinas, 2012; Soyata *et al.*, 2012; Rahimi *et al.*, 2012). Once the benefits of these frameworks became widely acknowledged by the research community, a new research trend appeared: MCC infrastructure planning (Gabner *et al.*, 2011; Park *et al.*, 2011; Pandey and Nepal, 2012; Oliveira *et al.*, 2013; Chen *et al.*, 2014). The scope of this field is to obtain an intelligent use of limited Cloudlet resources by applying sophisticated system evaluation techniques. Formal methods have been applied in diverse computer areas by evaluating system performance and assisting software engineers with

architecture planning. Most of them have dedicated to evolve what it is called Software Performance Engineering (SPE) (Herzog, 2001). SPE is a systematic, quantitative approach to constructing software systems that meet performance requirements, classified as real-time or responsive systems. SPE uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements. MCC has presented the need for applying SPE methods requiring to reach higher quality levels. For this reason, the current work focuses on MCC infrastructure planning applying SPE methods.

Evaluated Metric - One of the most important decisions when working with performance evaluation is the metric to observe. The papers that focus on Offloading Process Optimization have explored Execution Time and Energy Saving whereas the papers that investigate Infrastructure Planning have looked into Reliability, Availability, and Energy Saving. *Reliability* is defined as the probability that a device will perform its intended functions satisfactorily for a specified period of time under specified operating conditions (Araujo *et al.*, 2014). Since the performance of a system usually depends on the performance of its components, the reliability of the whole system is a function of the reliability of its components (Kuo and Zuo, 2003). *Availability* is defined as the probability that the system is operating properly at any given time (Oliveira *et al.*, 2013). Availability is the vital metric for nowadays systems; near 100% availability is becoming mandatory for both users and service providers. High availability is an important feature for MCC applications given that the cloud–dependency can introduce unexpected failures (Oliveira *et al.*, 2013). *Energy* and *Execution Time* are the most utilized metrics when evaluating the MCC systems. Computing speeds of mobile devices will not grow at the same pace as servers' performance (Nimmagadda *et al.*, 2010). This is due to several constraints, including: Form Factor—as users want devices that are smaller and thinner and yet with more computational capability; Power Consumption—insofar the current battery technology constrains the clock speed of processors, doubling the clock speed approximately octuples the power consumption. As a result of the above restrictions, it is difficult to offer long battery lifetimes with high clock speeds (Nimmagadda *et al.*, 2010). Therefore, execution time and energy will continue to be a MCC concern in long term, motivating further research under these topics. Although *Reliability*, *Availability*, and *Energy* are very important metrics, the current work is the only one that considers the *Execution Time* when modeling the MCC infrastructure. Response time becomes essential for mobile systems as far as it increases in complexity (Balan, 2006). In context-aware computing, for example, need to be analyzed in order to obtain real-time information

about a user's context (Hong and Landay, 2001). In many of these scenarios, the limited computing speed of mobile systems can be enhanced by offloading.

Modeling Granularity - Such aspect refers to the level of granularity the MCC architecture is represented — which MCC architecture parts are modeled. The offloading frameworks that invested in offloading process optimization did not use modeling. Among those papers that have applied modeling most of them did not consider the application as part of modeling strategy, it is, the SPNs, RBDs or Markov Chains did not include the behavior and structure of the application (only infrastructure components). Figure 3.1 depicts an RBD representation of one of the related work (Matos *et al.*, 2015). The model present the application itself as part of the representation but not partitioned in subparts. In our work the application source code is represented and evaluated with SPNs. Representing the source code enables the software engineer to access a more accurate result. Taking into account the availability metric, for example, it is obvious that the application may stop working prejudicing the availability in this way.

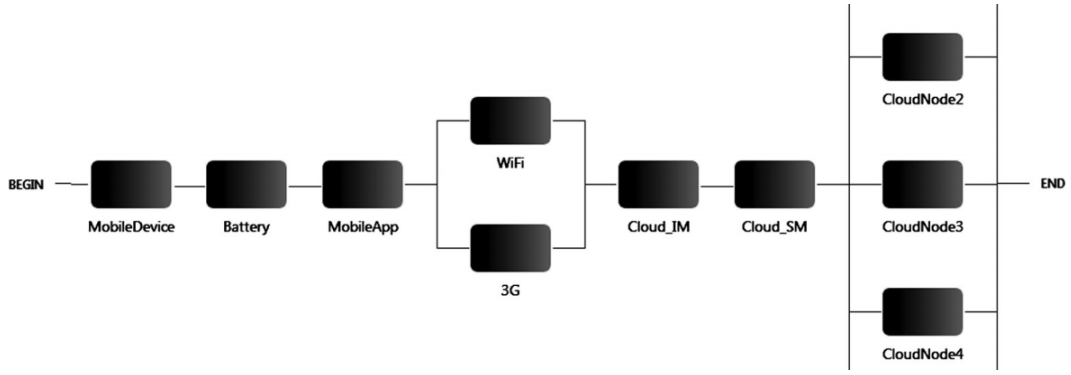


Figure 3.1: Infrastructure Components Model Example (Matos *et al.*, 2015).

Multiple Surrogates - The last column refers to the characteristic of modeling or not the target servers (or surrogates). The target servers are the machines where the offloaded tasks are processed. There are papers that models these machines, however, they do not represent the virtual machine. Besides, different from them, MCC-Adviser is able to tell how many servers are needed — something new in MCC field. Another new aspect, and not included at Table 3.1, is that MCC-Adviser generates and solves SPNs providing and automatic statistic report about performance.

3.2 MCC Offloading

Table 3.2 lists and classifies offloading systems according to three aspects: Offloading Metrics; Number of Targets; and Weighted Metrics.

- ***Offloading Metrics*** associate parameters used to calculate the offloading cost. A number of metrics could be used, but in this thesis part of our objective was to present the weighted metrics idea. Therefore, we do not include metrics such as the network instability or the memory load. Having said that, our proposal have used only CPU and RTT as offloading metrics together. However, only one paper (Abolfazli *et al.*, 2015) have applied both metrics concurrently, but without using balancing values.

- ***Offloding Targets*** are physical of virtual machines used to run the offloaded application workload. Four related work have used Multiple Targets and six have used Unique Target. It is important to stress that unique targets means in some situations the cloud is treated as an unique resource and the mobile client does not know the offloading targets. However, in most of the papers the experiments are performed by using one VM and not a bunch of VMs working as unique resource. In this thesis the offloading target are known aiming to realize how to optimize their use.

- ***Weighted Metrics*** are our main contribution regarding workload distribution in MCC. Only our work have employed such a functionality. Embracing more metrics in future work is essential to turn the algorithm more reliable and sensible, but for now the idea has shown to be a viable alternative. We are aware that the proposed weight values may not be the most efficient choice, since the range of options were not very high. However, again, the goal was not to find a static and strictly optimized weigh balance but showing the feasibility of reaching better results by varying those values.

Table 3.1: Related Work Comparison - MCC Modeling.

Related Work	Objetive	Evaluted Metric	Modeling Granularity	Multiple Surrogates
Kristensen (2010)	Offloading Process Optimization	Execution Time	-	Yes
Cuervo *et al.* (2010)	Offloading Process Optimization	Execution Time and Energy	-	No
Kemp *et al.* (2012)	Offloading Process Optimization	Execution Time	-	No
Kosta and Aucinas (2012)	Offloading Process Optimization	Execution Time and Energy	-	Yes
Soyata *et al.* (2012)	Offloading Process Optimization	Execution Time	-	Yes
Rahimi *et al.* (2012)	Offloading Process Optimization	Execution Time and Energy	-	Yes
Gordon *et al.* (2012)	Offloading Process Optimization	Execution Time and Energy	-	No
Pitkänen *et al.* (2012)	Offloading Process Optimization	Execution Time	-	No
Ou *et al.* (2007)	MCC Infrastructure Planning	Reliability	Infrastructure Components	No
Gabner *et al.* (2011)	MCC Infrastructure Planning	Reliability	Infrastructure Components	No
Park *et al.* (2011)	MCC Infrastructure Planning	Reliability	Infrastructure Components	No
Pandey and Nepal (2012)	MCC Infrastructure Planning	Availability	Infrastructure Components	Yes
Oliveira *et al.* (2013)	MCC Infrastructure Planning	Availability and Energy	Infrastructure Components	No
Chen *et al.* (2014)	MCC Infrastructure Planning	Energy	Infrastructure Components	No
Araujo *et al.* (2014)	MCC Infrastructure Planning	Reliability, Availability and Energy	Infrastructure Components	No
Matos *et al.* (2015)	MCC Infrastructure Planning	Availability	Infrastructure Components	No
Mendonca *et al.* (2015)	MCC Infrastructure Planning	Energy	Infrastructure Components	No
MCC-Adviser	MCC Infrastructure Planning	Execution Time	Application Partitions and Infrastructure Components	Applies Tasks Distribution and Estimates the Number of Needed Target Machines

Table 3.2: Related Work Comparison - Optimizing MCC Offloading Process.

Related Work	Offloading Metrics		Number of Targets		Weighted Metrics
	CPU	RTT	Multiple Targets	Unique Target	
(Abolfazli *et al.*, 2015)	X	X	X		
(Chen *et al.*, 2015)				X	
(Khan *et al.*, 2015)				X	
(Lin *et al.*, 2015)				X	
(Chen, 2015)	X		X		
(Kosta and Aucinas, 2012)		**X**	X		
(Kemp *et al.*, 2012)				X	
(Soyata *et al.*, 2012)		**X**	X		
(Pitkänen *et al.*, 2012)				X	
(Kristensen, 2010)				X	
Our Work	**X**	**X**	**X**		**X**

4

Evaluating MCC Applications

Evaluating the MCC infrastructure in a deep level of detail may provide to software engineers precise information. This section presents a way for representing method-calls and evaluating the MCC applications by using stochastic petri nets (SPNs).

4.1 Proposal Overview

Figure 4.1 presents, for example, how to calculate the number of needed machines for attending an expected performance. System Modeling (Costa *et al.*, 2015) is one way for evaluating a system based on probability, without having the infrastructure available. In other words, Stochastic Modeling makes it possible to calculate minimal requirements for infrastructure resources. We propose to adopt Stochastic Modeling for representing and evaluating the MCC infrastructure, divided into four activities. The activities are detailed following:

1. **Stochastic Modeling** - This activity intends to represent the MCC environment focusing on the application structure. Initially, the user needs to represent the system through stochastic models. In our approach, Stochastic Petri Net models are used for representing the application at source code granularity. Therefore, the application source code is required. Such a level of granularity (instead of only modeling machines where the processing occurs) enables the software engineer to obtain more accurate results. At this stage, the designer must also define the metrics of interest. In this work the metrics are execution time and energy. This part of the process resides one of the main research contributions of this thesis. Different from previous work, the models represent both, the application and the infrastructure with same model. The model permits to test distinct configurations

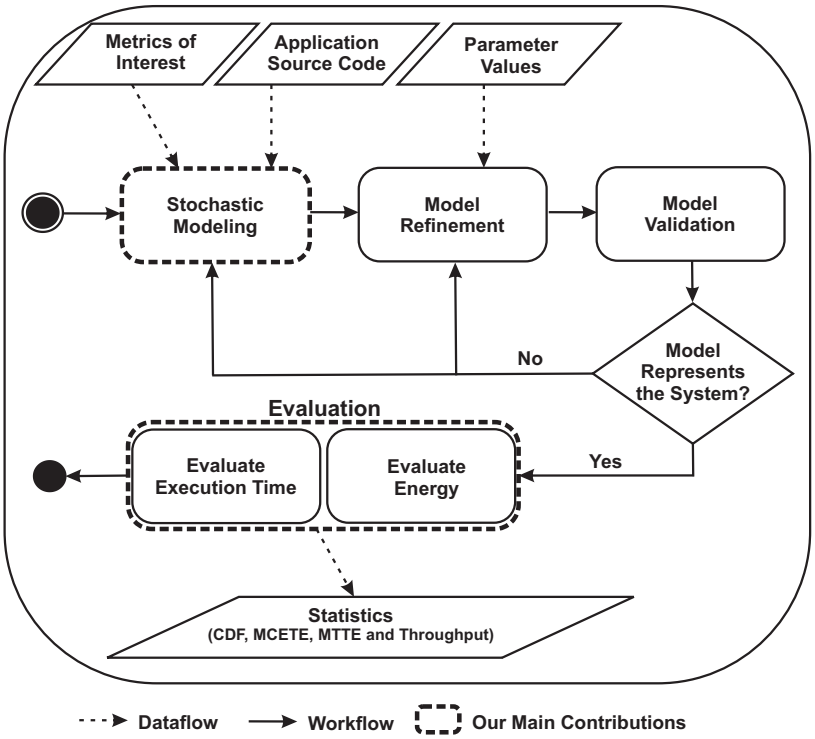


Figure 4.1: Evaluating and Planning MCC Infrastructure - An Overview

by simply changing some model parameters — no additional experiments are required. Besides, we developed a mechanism for automatically generating models and evaluating the application with minimal effort.

2. **Model Refinement** - At this activity the software engineer basically configure the model with specific values from real experiments. This process is called model refinement and the parameter values must be inserted into the model to emulate the actual system operation. There are cases where these parameter values are extracted from the literature. However, in this work, the software engineer must execute one experiment for each application which intends to evaluate.

3. **Model Validation** - The designer must ensure that the model trustworthy represents the real system, otherwise the previous steps should be repeated; Model validation consists of comparing the results from the model with the results from real experiments. Such a comparison must follow some statistical criteria, and in this work we adopt the statistical technique called Bootstrap (González-Rodríguez *et al.*, 2012). In practice, the software engineer will not have to always validate the generated models. The validation that have been performed in this work provide

validity evidences.

4. **Evaluation** - At this stage the model is prepared for calculating the metrics of interest and predict the application performance. By solving the model we have statistics related to execution time and energy. Specifically, four statistics are computed: throughput (*Tp*), mean time to execute (*MTTE*), cumulative distribution function (*CDF*) and mean consumption energy to execute (*MCETE*), which are evidenced by graphs. These results guide the software engineer at planning the MCC infrastructure. This part resides one of the main contributions of this thesis because we automatically generates the aforementioned statistics.

4.2 Evaluating MCC Applications with SPNs

One of the main cloud computing features is the unlimited resources availability. However, in the IT industry point of view the cloud has limited resources. IT companies tend to seek for the most optimized cloud infrastructure to support its systems. In MCC is not different, the society has adopted mobile devices in large scale and the applications must be evaluated considering multiple scenarios. Such evaluation should be performed in all the system life cycle. MCC applications can be evaluated even before the final deployment, bringing important indicatives of its behavior. Quite often, applications' QoE could be satisfied with smaller clouds, with fewer VMs. Aiming to choose one infrastructure, engineers should first to analyze application requirements creating execution profiles.

Before creating execution profiles, the application structure must be studied. In MCC, method call partitioning is one intuitive approach, since mobile applications are inherently organized in methods (Chun *et al.*, 2011b). Besides, method calls can bring high granularity as long as the methods are uncoupled. For example, consider the following snippet of code in Figure 4.2. Intending to paralelize the execution of such a code the dependence of variable "a" impacts on the final decision.

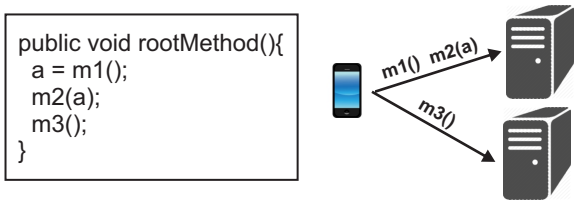


Figure 4.2: Method Call Partitioning Example.

Due to the variable "a", the calls "*m1()*" and "*m2(a)*" must be executed in sequence,

while "*m3()*" could be called in parallel. The developer would expect that distributing the execution of the *rootMethod()* on two machines (or CPU cores) would reduce the total execution time instead of using one resource. In this work, we present an approach to predict the application behavior. For that aim, some assumptions are needed: (i) the mobile device has an available high speed wireless network and is able to offload method calls to a nearby cloud; (ii) the target machines have similar resource configuration (CPU, Memory, etc) and similar current resource consumption.

Nonetheless, factors such as lack of resources, network instability, and allocation decision instructiveness may degrade the performance of the offloading process. These factors hamper the infrastructure planning since the application behavior is not easy to predict. Formal models, such as SPNs and Markov Chains, have been widely adopted for infrastructure planning (Araujo *et al.*, 2014; Campos and Silva, 2015a,b; Callou *et al.*, 2014; Haverkort, 2002). However, to the best of our knowledge, only a small number adopts formal models for MCC (Gabner *et al.*, 2011; Ou *et al.*, 2007; Matos *et al.*, 2015; Oliveira *et al.*, 2013; Araujo *et al.*, 2014), and no work consider the application at source code level.

Energy consumption estimation is also an essential issue on the development life cycle of mobile applications. Without loss of generality, there are two basic approaches based on simulation for estimating software energy consumption: (i) instruction based simulation and (ii) hardware based simulation (N. Nikolaidis, 2002). In hardware simulation, despite the great computational effort, more accurate results might be obtained in comparison with instruction simulation due to the laborious system specification. However, instruction simulation has been adopted by many works in order to provide energy consumption estimation in a satisfactory period of time. Some works concern hardware and instruction simulation. However, to the best of our knowledge, no one uses formal models for evaluating MCC focusing on energy consumption and the application source code.

This work proposes an approach that provides statistic information about the mobile application behavior, representing method calls dependency with stochastic models. More precisely, this work seeks to answer the following five questions:

i) How many VMs are needed to satisfy an application's required average execution time constraint?

ii) What is the number of method calls per time unit (throughput)?

iii) What is the mean time to execute one application when using a certain number of VMs?

iv) What is the mean energy consumed by the mobile device to execute one application when using a certain number of VMs as target resources?

v) What is the probability of finishing the application execution by a specific time when using a certain number of VMs?

We answer these questions by providing a way of evaluating the mobile cloud through SPNs. First, a static code analysis is carried out and SPNs are automatically generated based on such code. Then, the SPNs are employed to evaluate the performance of the method call execution. As an example, Figure 4.3 depicts the SPN representation for the basic structure of a method call.

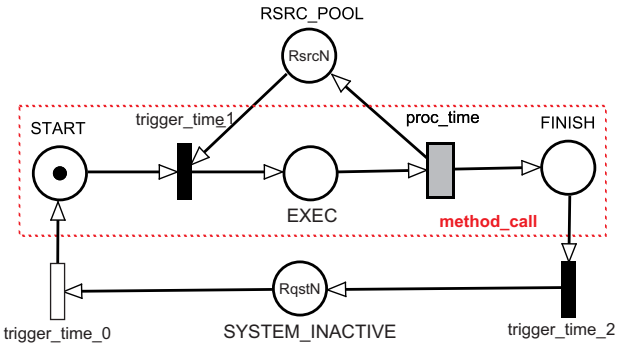


Figure 4.3: SPN Representation of One Application with Only One Method Call without Absorbing State.

The SPN method call is composed of four transitions. The first transition (*trigger_time_1*) is immediate, associating zero. The second transition (*proc_time*) is called General Time High-level Transition because when the SPN is generated, no probability is assigned to it. This transition is depicted by a gray rectangle and the model is later refined by assigning the respective distribution parameter values. The other two immediate transitions (*trigger_time_0* and *trigger_time_2*) are needed to enable the model to return to the initial state when the execution finishes.

The SPN model comprises four places. The place START, when containing a token, means that the workload is able to be processed. The place *EXEC* represents the phase when the method started its execution by allocating one resource, and thus, decreasing the number of markings at the place *RSRC_POOL*. Therefore, the number of tokens present at the place *RSRC_POOL* represents the current available Resources Number, *RN* (e.g., the number of VMs or phisical machines).

The pool of resources is a powerful mechanism — just changing its marking number allows different scenarios to be analyzed. The delay associated with the transition *proc_-*

time represents the average method processing time. The place *FINISH* represents that the method call has completed and is available for further calls. Finally, when the place *SYSTEM_INACTIVE* has a marking (i.e., a token), this indicates that there are no method calls running at the moment and that the system is idle. Such SPN pattern can be extended to evidence the method calls data dependency of any application. The pattern embraces general features common to concurrent systems.

We have designed and implemented a tool called MCC-Adviser that can assist software engineers with planning mobile cloud infrastructure. MCC-Adviser is based on the Mercury engine (Callou *et al.*, 2014; Silva *et al.*, 2013) and supports the analysis of mobile applications from different perspectives, being able to automatically compute four metrics: throughput (*Tp*), mean time to execute (*MTTE*), cumulative distribution function (*CDF*) and mean consumption energy to execute (*MCETE*), which are evidenced by graphs. These metrics are calculated by numerically solving CTMCs or by simulation. The main problem of analytical evaluation methods is the state space. Real application systems usually generate huge state spaces. In some situations the simulation is the only feasible approach for performance evaluation (Balbo and Chiola, 1989; Silva *et al.*, 2013; German *et al.*, 1995).

4.2.1 Throughput

The throughput (Tp_n) represents how many method-calls per unit time one application can execute when offloaded to the cloud. This metric is obtained based on Equation 4.2.1. (Maciel *et al.*, 2011). Tp_n is obtained by computing the expected value of tokens at a place, multiplied by the inverse of the transition delay. Such a transition delay ($Time_n$) corresponds to the communication time, it is, strictly the time taken to send and receive bytes.

For calculating the throughput there are two possibilities, considering a Single Server Semantics (SSS) or an Infinite Server Semantics (ISS). In this work both ways can be used (SSS or ISS). In the SSS the flow of tokens will occur in series, regardless of the degree of the transition activation. In the ISS, every set of tokens of the enabled transition is processed simultaneously. Equation $Tp = \left(\sum_{i=1}^{Z} P(m(EXEC_n) = i) \times i\right) \times \frac{1}{Time_n}$ calculates the throughput according to the ISS strategy, and Equation $Tp = P(m(EXEC_n) >= i) \times \frac{1}{Time_n}$ for SSS. The variable i represents the weight of the arc that links the place $EXEC_n$ and the subsequent transition (refined by $Time_n$ value). The variable i may vary until Z, where Z is the highest enabling degree of the subsequent transition at the place marking $m(EXECn) = i$.

4.2.2 Execution Time (MTTE and CDF)

The MTTE (Mean Time To Execute) represents the average time one application takes to finish its execution. To compute such measure, the SPN must be slightly different to the presented before. Figure 4.4 presents the SPN for computing MTTE. The SPN now presents one new possible state called *absorbing state* (Bolch *et al.*, 2006). The state is called absorbing if it is impossible to leave it (i.e., Pii = 1). The MTTE is the expected time to reach a deadlock marking (Nelson, 2013). The MTTE is based on a set of probability estimations for when one token goes from the *START* place to the *FINISH* place.

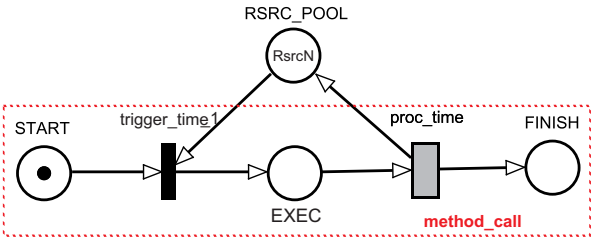


Figure 4.4: Basic SPN Representation of One Application with Only One Method Call Using Absorbing State.

Cumulative Distribution Functions (CDF) can be calculated based on SPNs (Trivedi, 2002). Figure 4.5 depicts an example of a *CDF* varying the number of available resources (VMs). Based on CDFs, the probability of finishing execution can be obtained to specific periods of time. Calculating the probability of finishing an program execution before a specific time $[P(T < t)]$ and the probability of finishing execution in a time interval $[P(t_1 < T < t_2) = P(T < t_2) - P(T \leq t_1)]$, for instance.

4.2.3 Energy

Consider the model in Figure 4.6, which represents three method-calls. Each method-call have an individual throughput and contributes for spending energy. One of the main causes for spending energy is transferring method-calls (with data) through the Internet (da Silva *et al.*, 2014). It is possible predicting the mean energy consumption using SPNs in such a scenario by calculating the throughputs, and required energy for transferring data.

The Mean Consumed Energy to Execute (MCETE) is the expected energy consumption of an application execution. As far as the SPN model represents a set of K

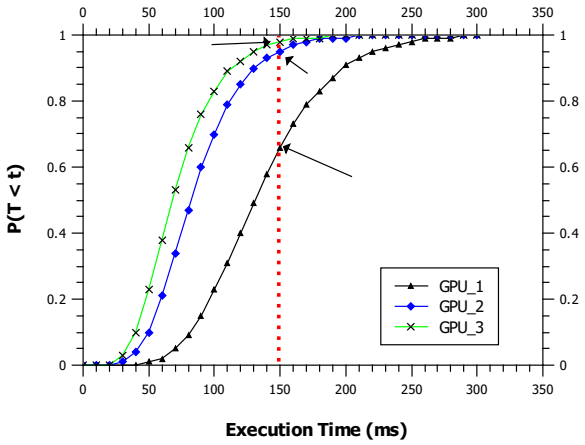


Figure 4.5: Example of *CDF* based on SPN.

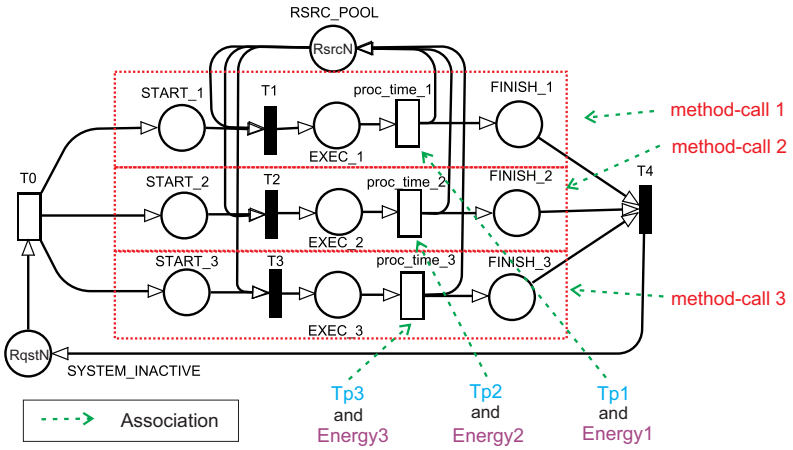


Figure 4.6: Energy and Power Scheme.

method-calls, then individual measurements for each method-call must be considered. Equations $\mathbf{MCETE} = TTime \times \sum_{n=1}^{K} (Tp_n \times Energy_n)$.

$TTime$, represents the total time in which the application was executed by using one resource (VM or physical machine). The $TTime$ is multiplied by the sum of the products between throughputs (e.g.: Tp1, Tp2 and Tp3) and energy consumptions (e.g.: Energy1, Energy2 and Energy3). It is important to note that the throughput multiplied by the energy of a method-call is equivalent to the mean Power for that method-call.

$Energy_n$ means the energy consumption resulted from offloading a method-call. $Energy_n$ can be obtained by multiplying two values: $EnergyPerByte$ and $BytesNumber_n$, as presented in Equation $Energy_n = EnergyPerByte \times BytesNumber_n$.

$EnergyPerByte$ means the energy necessary for transferring one byte between the mobile device and the cloud. $EnergyPerByte$ was obtained through experiments and

will be presented later. $BytesNumber_n$ means how many bytes were offloaded for each method-call. In this work, we consider that most of the processing (for n method-calls) occurs at the cloud (see Figure 4.7). All those methods that do not depend on physical mobile device resources (e.g.: GPS) are able to be offloaded. Therefore, the energy consumption is approximately equivalent to the effort in offloading bytes.

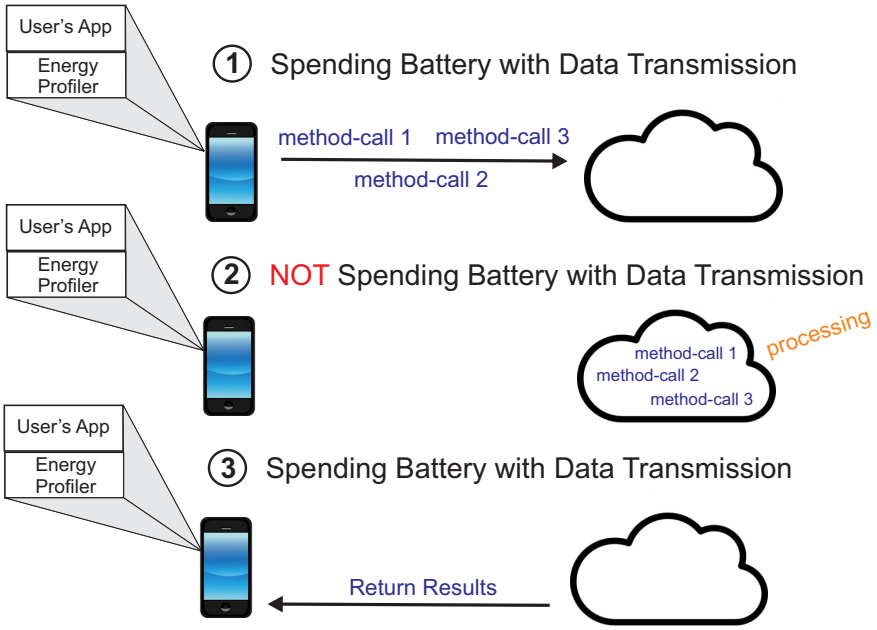


Figure 4.7: Energy Profiling Scheme.

It is important to highlight that the bytes considered in this work are those data involved with the offloading process. The bytes include bytes that will be processed and bytes generated by offloading instructions. Using a music player application as an example, a streamed song itself with commands (e.g.: play/stop) represents the bytes we are interested. All these data transmission results in spending mobile device battery. Therefore, it is important to cautiously profile the mobile applications for computing the MCETE. In other words, the programmer must know how much data the application is offloading with the maximum possible accuracy.

Power (P) is defined as the rate at which work is done upon an object. Like all rate quantities, power is a time-based quantity. Power is related to how fast a job is done. Two identical jobs or tasks can be done at different rates - one slowly or and one rapidly. The work, which in this document means energy (E), is the same in each case (since they are identical jobs) but the power is different. Equation $Pi(t) = \frac{E(t)}{t}$ presents how to obtain the instantaneous power (Pi).

During the execution of an application, the average power ($\widetilde{P}$) over the period of time t also can be obtained by

$$\widetilde{P} = \frac{\int_0^t E(t)dt}{\Delta t} = \frac{\int_0^t E(t)dt}{t-0} = \frac{\int_0^t E(t)dt}{t},$$

As

$$\int_0^t E(t)dt = Etotal = \widetilde{P} \times t,$$

then, as Figure 4.8 illustrates, $Etotal$ and $\widetilde{P}$ may be calculated. $Etotal$ is represented by the area below the line of the power graph and $\widetilde{P}$ is computed by summing up the n instantaneous power and dividing by n:

$$\widetilde{P} = \frac{\sum_1^n Pi(t)}{n}$$

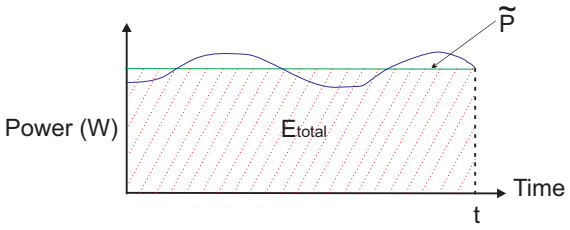


Figure 4.8: Energy and Power Scheme.

The unit for standard metric energy is the Joule and the standard metric unit for time is the second, so the standard metric unit for power is a Joule/second, defined as a Watt and abbreviated W. In this work the adopted unit for energy is milijoules (mJ), a common unit for energy in mobile devices.

4.3 MCC-Adviser: An Evaluation Assistant

We have implemented MCC-Adviser, an evaluation tool for MCC applications. MCC-Adviser is implemented in Java and aims at assisting software engineers planning MCC environments. Figure 4.9 illustrates an overview of how MCC-Adviser works. The first step is to execute an experiment using one machine (VM or physical server) to process the mobile device requests. The objective is to collect data needed to refine the generated model. MCC-Adviser requires:

- the source code (to build the SPN structure),

- the total communication time,
- the communication time delay (per method-call),
- the number of transfered bytes (per method-call), and
- the “energy spent per byte".

The MCC-Adviser, then, provide measures and plots describing performance and energy profile according to different number of VMs. Following, these steps are detailed.

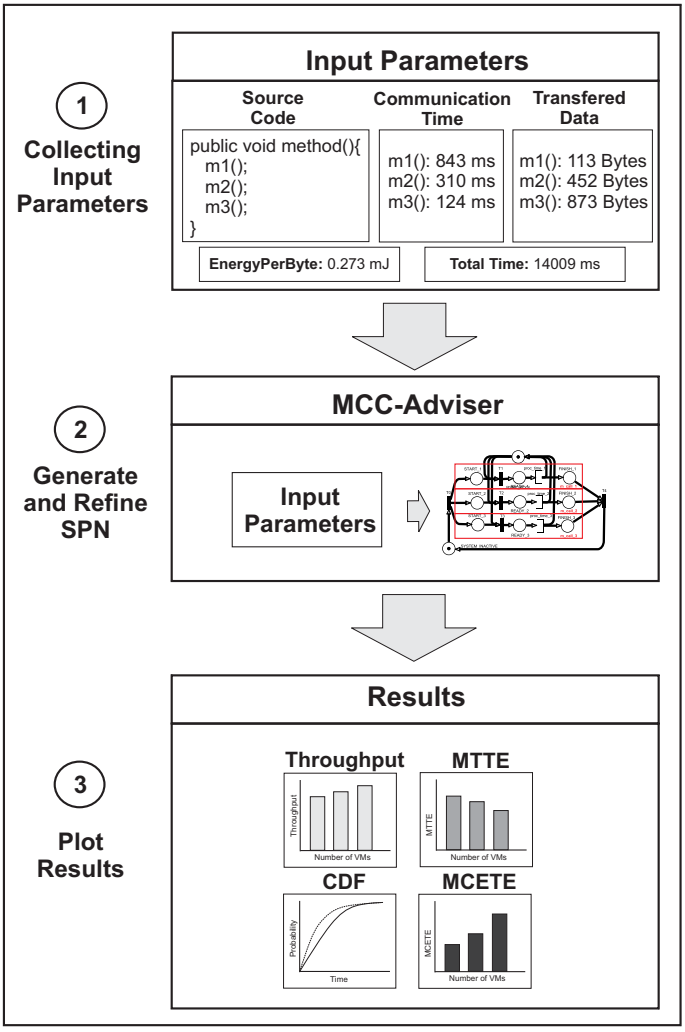


Figure 4.9: MCC-Adviser Overview.

4.3.1 Collecting Input Parameters

Aiming to refine the SPN, input parameters are strictly necessary. Figure 4.10 presents an overview of the *Collecting Input Parameters* phase. The parameters of interest are basically two: time and transfered data. As far as the MCC-Adviser plays the role of an adviser, it is expected that the software engineer executes as minimal as possible tasks to get the estimated statistics. Three softwares were developed aiming to assist with such tasks: *MCC-Instrumenter*, *MCC-Logger-Client* and *MCC-Logger-Server*.

As Figure 4.10 illustrates, in Step 1, *MCC-Instrumenter* receives the original source code (root method) as input and generates an instrumented code. In practice, the user uploads an entire Java class indicating which method she wants to instrument. Then, *MCC-Instrumenter* instruments the code by adding directives before and after each method-call. *MCC-Instrumenter* is supported by the library BCEL[1].

The process of collecting the metric time must be performed at both sides, the mobile and the cloud. *MCC-Instrumenter* instruments the user's code with the class *TimeLogger* (see Code 4.1). In other words, *TimeLogger* accompanies the instrumented code. The methods *registerStart()* and *registerEnd()* are inserted into the code. After that, MCC-Adviser returns a zip file containing the instrumented class, and auxiliary classes.

```
1  public class TimeLogger{
2    public static double initTime = 0;
3    ManageFile m = new ManageFile ("log-exec-time.txt");
4
5    public static void registerStart(){
6      initTime = System.currentTimeMillis();
7    }
8
9    public static void registerFinish(String methodCallName, int line){
10     m.WriteFile(methodCallName+"["+line+"]:"+(System.currentTimeMillis() - initTime));
11   }
12 }
```

Code 4.1: TimeLogger Class

More specifically, related to time, *MCC-Instrumenter* generates logs addressing the following metrics: the total execution time, the communication time, the execution time in the cloud. The communication time delay is obtained by the difference between total execution time and the execution time in the cloud.

Still observing Figure 4.10, in Step 2, the programmer will execute the instrumented source code and collect the input parameters. At the mobile device side, the programmer needs to install the instrumented application and the *MCC-Logger-Client* application.

[1]BCEL: https://commons.apache.org/proper/commons-bcel/

MCC-Logger-Client monitors the instrumented application and logs the communication time delays and transfered bytes — for each method-call. These logs are stored at a remote server to be used by the MCC-Adviser afterwards. At the server side, the original code executes the requested tasks from the mobile device. The *MCC-Logger-Client* is an evolution of the (PowerTutor, 2014) PowerTutor open-source application. *MCC-Logger-Server*, in turn, is responsible for receiving the requests and properly executing the specific method-calls. For that aim, *MCC-Logger-Server* uses Java Reflection for identifying the method-calls targets.

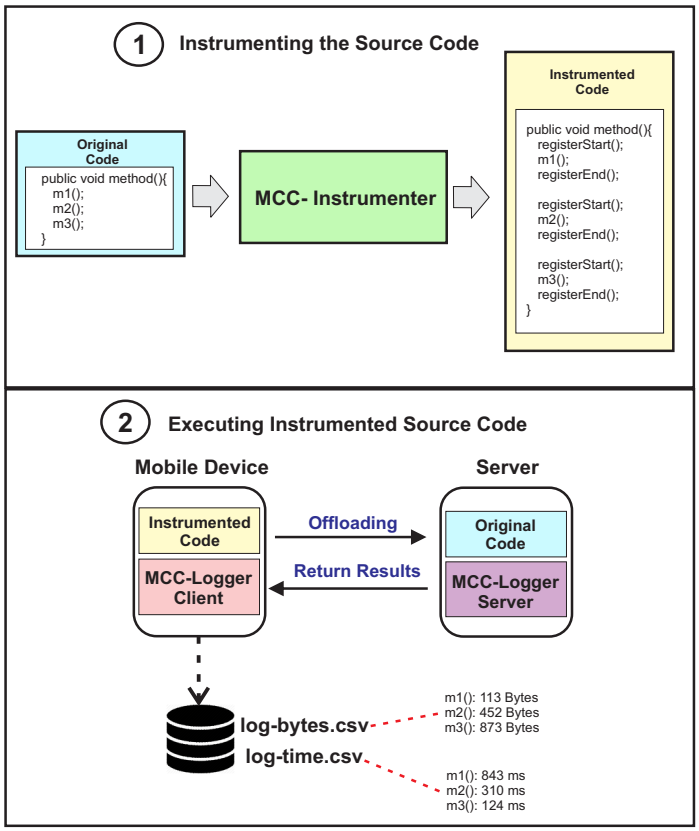


Figure 4.10: Collecting Input Parameters.

As infrastructure, the private cloud Eucalyptus 3.4.0.1 (Nurmi *et al.*, 2009) was used with two physical machines (one node and one controller). The physical machines have the following configuration: Intel Core i7-3770 3.4 GHz CPU, 4 GB of RAM DDR3, and 500 GB SATA HD. An Ethernet network is adopted to connect the physical servers through a single switch and one VM of type *m1.medium* (1 CPU, 512MB of RAM, and 10GB Disk). At the mobile device side, a Samsung Galaxy Note 4 was used running the Android version 5.1.1 Lollipop. Only the essential process was running at the device

during this experiment.

4.3.2 Solving SPNs and Plotting Results

After collecting and informing the input parameters to MCC-Adviser, the stochastic model is generated based on the source code structure. The act of configuring transitions with input parameters is called refinement process. Only when setting transitions and places with initial values, the stochastic model may generate some result. Therefore, the final step is to solve the SPN model and graphically generate the graphics.

MCC-Adviser is based on the Mercury engine (Callou *et al.*, 2014) (Silva *et al.*, 2013). Mercury was developed by MODCS[2] research group to allow the evaluation of performance and dependability models. The proposed environment can be adopted as a modeling tool for a number of formalisms but MCC-Adviser uses the Mercury API to evaluate only SPNs. Mercury can be used as a dashboard through a Graphical User Interface (as illustrated in Figure 4.11) or working as an API. As an API, Mercury provides one class called *SPNModel* that represents one SPN. Thus, the first step MCC-Adviser does is to instantiate the *SPNModel* class. Next, for each method-call, the *Places*, *Transitions* and corresponding *Arcs* are instantiated and incorporated into the model.

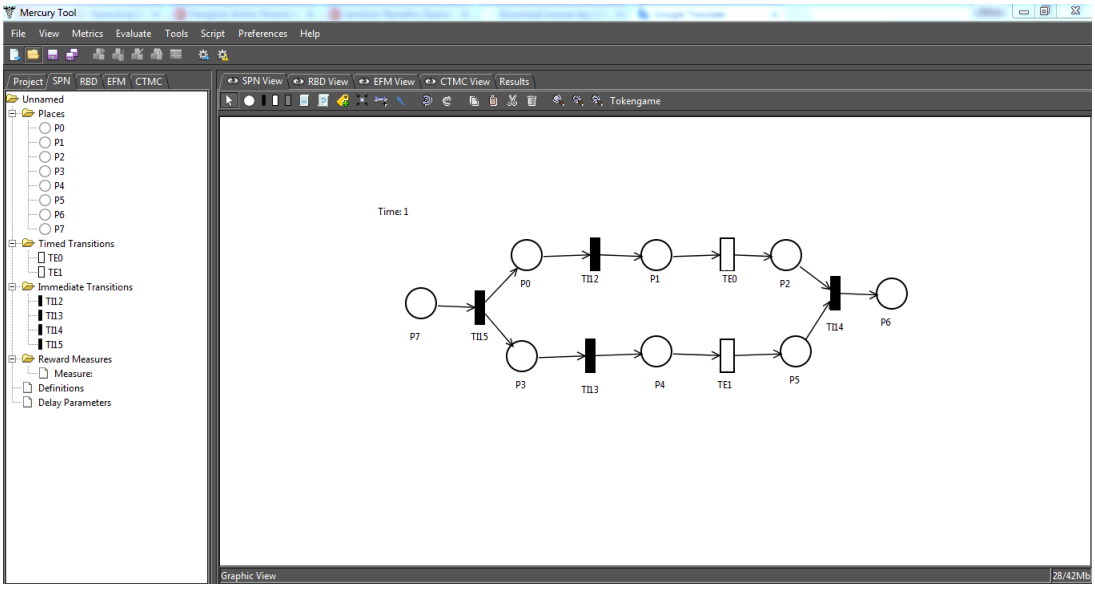


Figure 4.11: Mercury GUI

In practice, MCC-Adviser is deployed as a jar file called *mcc-adv.jar*, containing all dependencies, including the Mercury library. Figure 4.13 describes a sequence diagram

[2]MODCS: http://www.modcs.org/

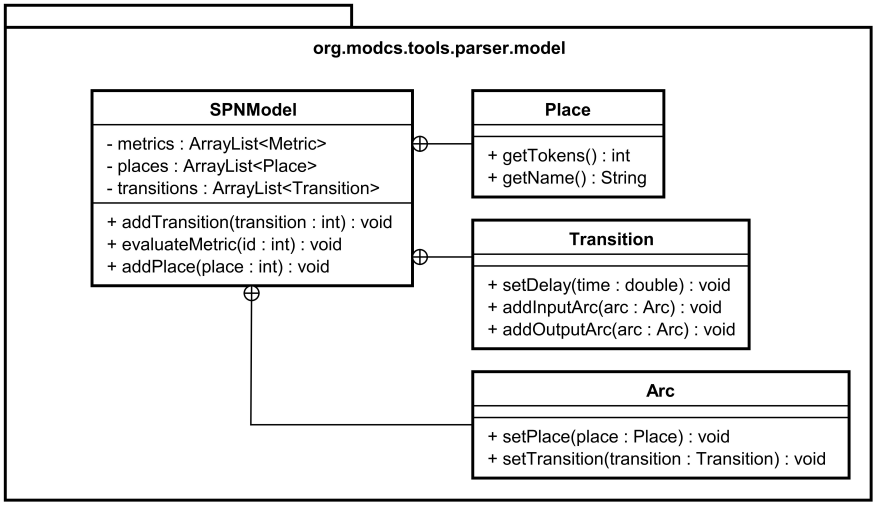


Figure 4.12: Main Classes of Mercury API

for MCC-Adviser behaviour. The figure highlights the communication between MCC-Adviser core and the Mercury library. First, the user informs the application source code, indicating the target class and its desired root method. Next, the MCC-Adviser statically analyzes the code and generates a method-call dependencies tree. Calling Mercury, the MCC-Adviser then instantiates transitions, places and arcs, creating an SPNs. Next, the MCC-Adviser enable the user to inform the input parameters that are necessary to refine the model and compute metrics. Finally, the MCC-Adviser generates graphs.

We also have implemented a graphical user interface, which requires minimum user intervention, hiding all the modeling complexity of MCC-Adviser. At the webpage the user may download the desktop version or try one of the following functionalities online: Code Instrumentation, Energy Evaluation, and Time Evaluation. To use the modeling prototype, the engineer should inform all needed input parameters. In background, the MCC-Adviser generates the required statistics.

4.3.3 Web Application Prototype

In this work we have implemented a web application prototype where users may download and test MCC-Adviser through the link: *http://cin.ufpe.br/∼faps/mcc-adv/*. Figure 4.14 depicts the first page. In this page the user have three options. First, she may download a Desktop version, with corresponding user manual. In the other two buttons, the user may test — online — the time and energy evaluation.

Figure 4.15 shows the step one of the online time evaluation. At this step the user should insert a piece of Java code for static analysis.

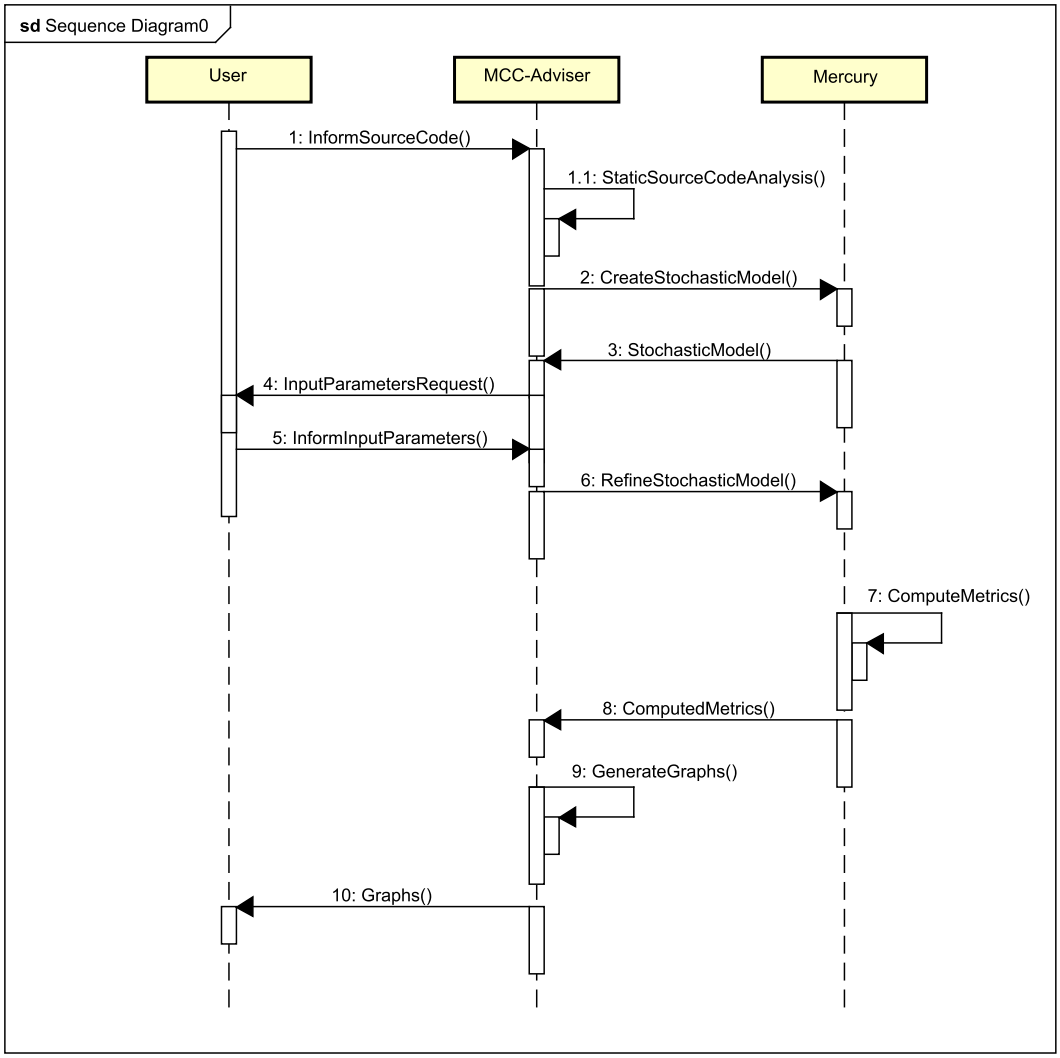


Figure 4.13: MCC-Adviser Sequence Diagram.

In step two (see Figure 4.16), the user should insert the time for each one of the method-calls.

Finally, in step three (see Figure 4.17), MCC-Adviser presents the statistic report containing the Throughput, MTTE and CDF. Besides, the usee may obtain the exact probabilities for finishing the application by simple informing the desired time interval.

4.4 Experiment for Estimating the “EnergyPerByte"

This section presents how we have estimated the average energy consumption for transferring one byte (*EnergyPerByte*) involving the mobile device and the cloud. Different from the metric time, the task of recording energy encompasses diverse challenges (Li *et al.*,

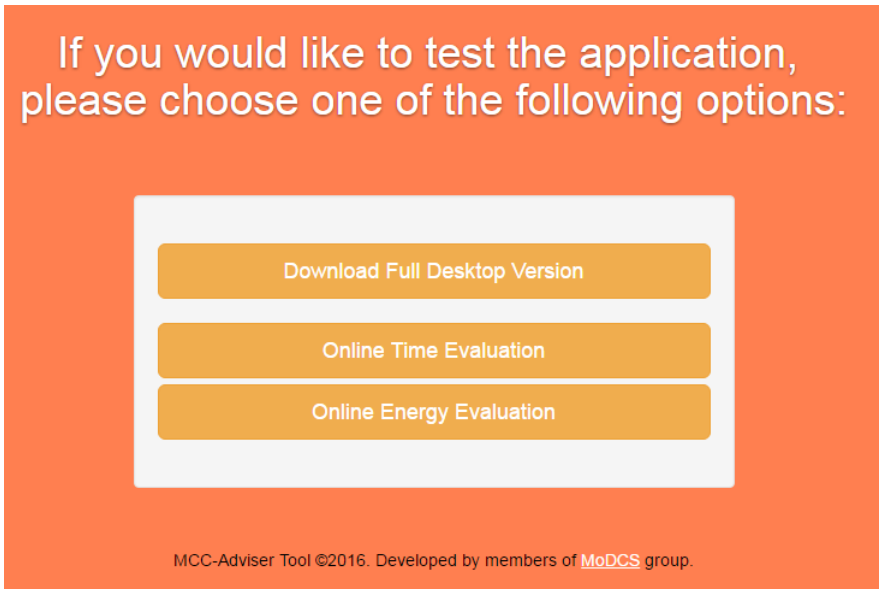


Figure 4.14: MCC-Adviser Web Application - First Page

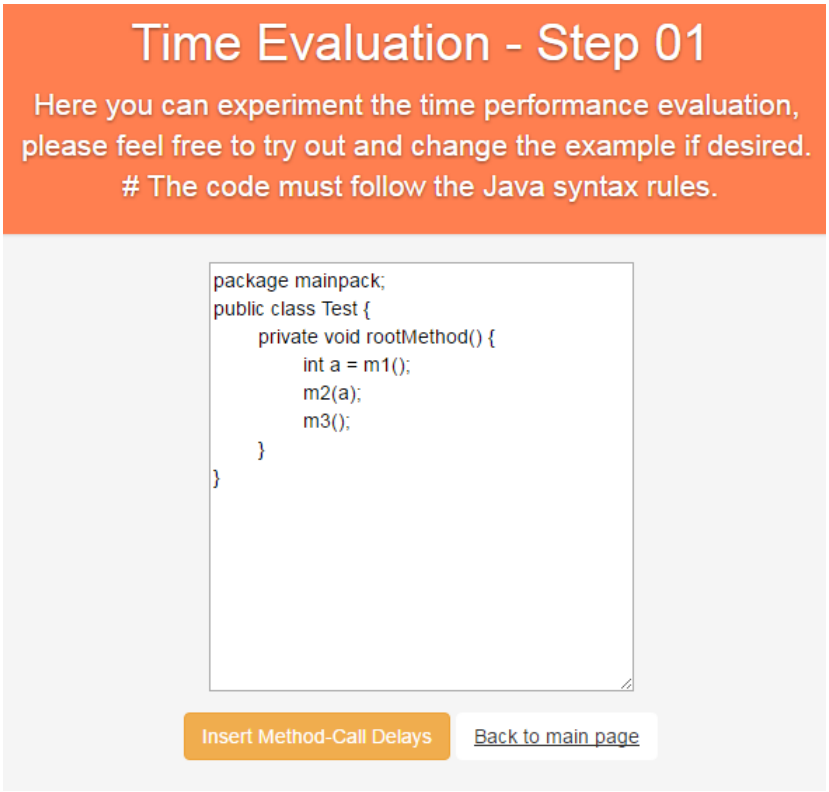


Figure 4.15: MCC-Adviser Web Application - First Step.

2013; da Silva *et al.*, 2014; Oliveira *et al.*, 2013; Araujo *et al.*, 2016; Silva *et al.*, 2014a; Callou *et al.*, 2011; Tavares *et al.*, 2010; Callou *et al.*, 2008; Tavares *et al.*, 2007; Junior *et al.*, 2006) related to hardware and runtime system when measuring energy consumption

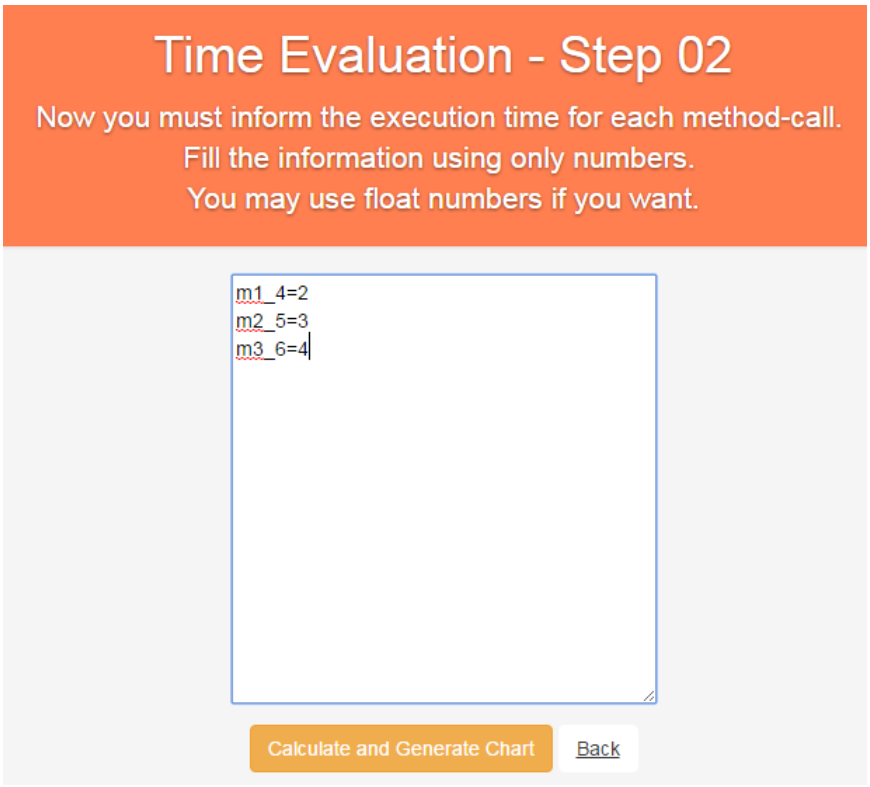


Figure 4.16: MCC-Adviser Web Application - Second Step.

of constrained devices.

In terms of hardware, the main obstacle is the difference between the speed at which instructions execute and hardware devices can perform energy measurements. On modern processors, individual instructions will execute at a rate of several million per second. At best, power meters can sample electrical power draw at several tens of KHz, which means that each sample will include the power consumption of hundreds, perhaps thousands, of instructions.

The runtime system of an Android smartphone includes both the Android operating system and the Dalvik Virtual Machine. The runtime system implements several types of behaviors that affect energy consumption of an app, thread switching, garbage collection, and tail energy. However, the details of the duration, frequency, and timing of these events is, by design, hidden from the app. Although it would be straightforward to modify the runtime systems to track these events, this would introduce considerable overhead and reduce the portability of the approach, as it would be necessary to provide custom runtime systems for each smartphone platform.

There are hardware profilers (e.g.: Watts Up (WattsUp, 2016)) that only record the total consumed energy and not the individual energy per application. There are also some

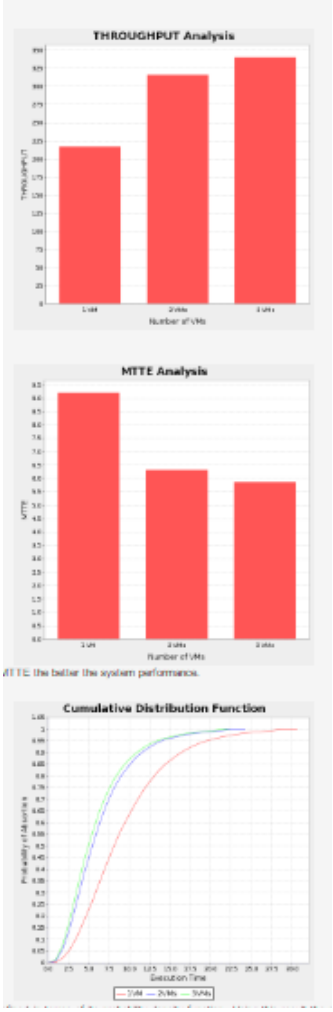


Figure 4.17: MCC-Adviser Web Application - Third Step.

profilers at software level (e.g.: eDoctor (Ma *et al.*, 2013) and PowerTutor (PowerTutor, 2014)) that record the energy per application. Therefore, this work have adopted this software approach.

Aiming to calculate the MCETE for transferring one byte (*EnergyPerByte*), it is necessary to profile one application during a certain period of time. Next, *EnergyPerByte* is obtained by dividing the total amount of energy (*TotalEnergy*) by the total number of transfered bytes (*TotalBytesNumber*), as shown in Equation $EnergyPerByte = \frac{TotalEnergy}{TotalBytesNumber}$.

Based on the (PowerTutor, 2014) PowerTutor application, the *MCC-Logger-Client* was also adapted for recording, besides bytes and time, the Power consumption. The objective was to capture the aforementioned metrics through a controlled experiment as illustrated in Figure 4.18.

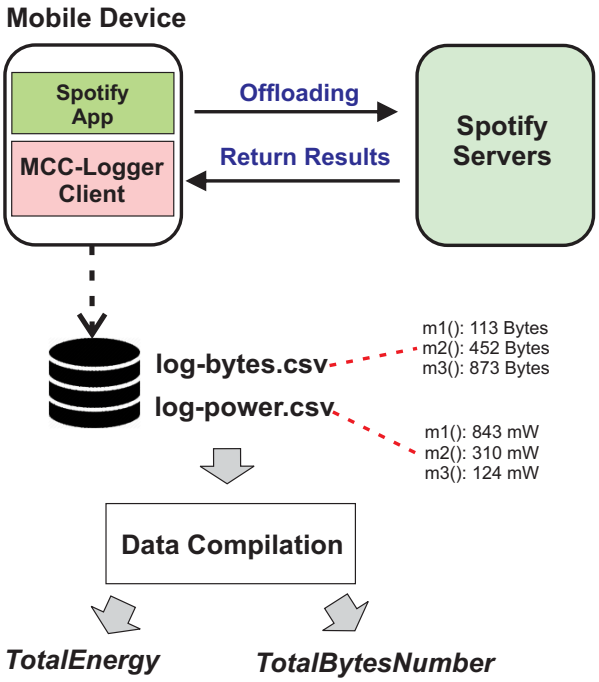
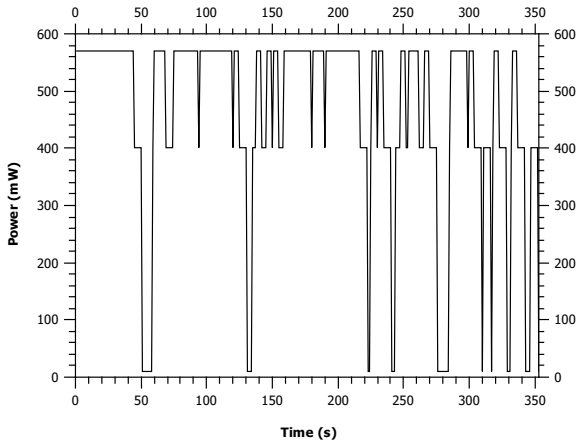


Figure 4.18: Architecture Scheme for Computing the *EnergyPerByte*.

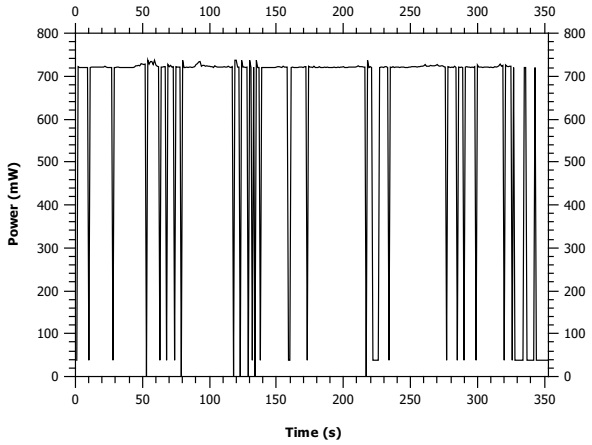
The main hardware components that may impact on energy consumption of mobile devices are: Display Backlight, GPS, CPU (Ardito *et al.*, 2013), and Connection type (3G, 4G, 5G, WiFi, etc); The Display and the GPS were not profiled because they remained disabled during the experiment. Considering that most of execution occurs at the cloud, the mobile device does not require much processing. Most of the processing is related to transferring bytes and therefore the connection channels (3G and WiFi) were the focus point.

The experiment was performed by profiling the mobile device power to calculate the consumed energy. The application Spotify[3] was adopted during the experiment. Spotify plays music online by streaming. Thus, Spotify was executed for six minutes . It is important to stress that there were other processes running on the device, including the operating system. All processes were also profiled, not only the Spotify application. However, all background services not required for running the operating system were disabled. Figures 4.19a and 4.19b present the profiled power over time for 3G and WiFi, respectively. The *TotalEnergy* for 3G and WiFi were calculate by estimating the area bellow the line graphs wheres the *TotalBytesNumber* were simply profiled and summed up.

[3]Spotify: https://www.spotify.com/br/



(a) Power for 3G Channel.



(b) Power for WiFi Channel.

Figure 4.19: Power over Time

Table 4.1 presents the *TotalEnergy*, *TotalBytesNumber* and *EnergyPerByte*. The energy for transferring one byte was about 0.273 mJ for 3G and 0.0024 mJ for WiFi. We believe that the energy for 3G was higher because its bandwidth is lower then WiFi, requiring more energy for transferring bytes.

Table 4.1: Consumed Energy for Offloading one Byte

TotalEnergy (mJ)		**TotalBytesNumber**		**EnergyPerByte (mJ)**	
3G	**WiFi**	**3G**	**WiFi**	**3G**	**WiFi**
164029	218355	600363	90819921	0.273	0.0024

Although the *TotalEnergy* using WiFi was higher than 3G — for the amount of bytes was the opposite. The flow of bytes for WiFi was 150% higher then using 3G.

Therefore, is “easier" to transmit bytes using WiFi. 3G requires 113 times more Energy for transmitting one byte then using WiFi connection.

4.5 Case Study One - Time Metric - Reduce Color Application

This section presents a case study observing the execution time metric using an application for reducing images color.

4.5.1 Model Presentation

As we have explained in the previous section, MCC-Adviser hides the complexity of the problem and the underlying SPN model from the end user, presenting only a User Interface and graphical prediction reports. However, to offer a full understanding of MCC-Adviser, we give a detailed description of all involved steps—from building the SPN to obtaining the results—in this section. This work does not consider applications that needs user interaction during the application execution, since it deals with offline evaluation.

We implement and analyze three versions (A, B, and C) of an image processing Android application following the principles of method call computation offloading (Kosta and Aucinas, 2012), (Cuervo *et al.*, 2010). The implementation uses a simple elastic client server architecture with Remote Method Invocation (RMI), but focusing on explaining the modeling evaluation. The relevant parts of the offloading source code are presented in Code 4.2. The depicted code arrangements only show the corresponding heaviest methods of the three application versions.

Figure 4.20 reveals the method calls distribution scheme of the three applications. The client class resides on the mobile device and makes image processing calls to the server by passing one or more inputs (original images). The client connects to one or more VMs and then calls the method *reduceColor* in the server side.

The method calls inside *Application_A* present dependencies by passing image inputs as method arguments (lines 5 to 7). In *Application_B*, there are two dependent method calls (lines 16 and 17) and one independent (line 19). The last application, *Application_C*, is dependency free.

```
1
2 public class Application_A{
3  public List<Image> reduceColorClient(Image image){
```

```
4    List<Image> results = new ArrayList<Image>();
5    Image image2 = reduceColor(image);  /* m_call_1 */
6    Image image3 = reduceColor(image2); /* m_call_2 */
7    Image image4 = reduceColor(image3); /* m_call_3 */
8    results.add(image4);
9    return results;
10  }
11 }
12
13 public class Application_B{
14  public List<Image> reduceColorClient(Image image1, Image image2){
15   List<Image> results = new ArrayList<Image>();
16   Image image3 = reduceColor(image1); /* m_call_1 */
17   Image image4 = reduceColor(image3); /* m_call_2 */
18   results.add(image4);
19   results.add(reduceColor(image2));   /* m_call_3 */
20   return results;
21  }
22 }
23
24 public class Application_C{
25  public List<Image> reduceColorClient(Image image1, Image image2, Image image3){
26   List<Image> results = new ArrayList<Image>();
27   results.add(reduceColor(image1));  /* m_call_1 */
28   results.add(reduceColor(image2));  /* m_call_2 */
29   results.add(reduceColor(image3));  /* m_call_3 */
30   return results;
31  }
32 }
33
34 public class Server{
35  public Image reduceColor(Image image) {
36   // JavaCV code supressed
37  }
38 }
```

Code 4.2: Client and Server Classes—Image Processing Source Code.

The server side adopts the Open Source Computer Vision Library (OpenCV, 2015) and one Java wrapper called *JavaCV* (JavaCV, 2015). We implement the computing vision example of Picture's Colour Reduction (Reduction, 2015), in which images are transformed by decreasing the number of colors depending on the picture's size. Such an activity may be quite time consuming. The test-bed was composed of a private cloud comprising four machines with the same hardware configuration: Intel Core i7-3770 3.4 GHz CPU, 4 GB of RAM DDR3, and 500 GB SATA HD. One machine is configured as the front-end while the remaining three are processing nodes. The Linux CentOS 6 (cen, 2015) operating system and Eucalyptus platform 3.4.0.1 (Nurmi *et al.*, 2009) are adopted. An Ethernet network is adopted to connect the PCs through a single switch and VMs of

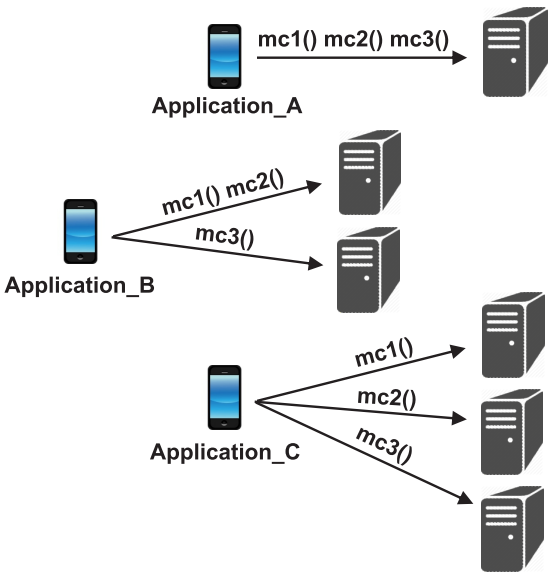
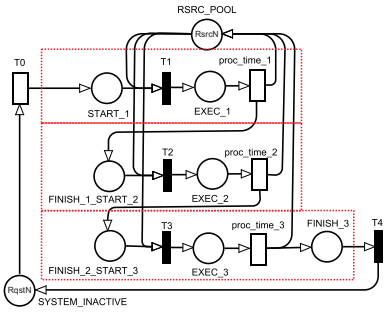


Figure 4.20: Method Call Distribution Obeying Code Dependency Constraints.

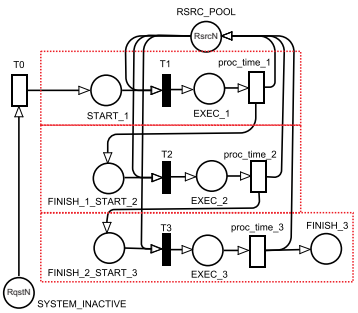
type *m1.medium* (1 CPU, 512MB of RAM, and 10GB Disk).

We have designed and represent high-level SPN models of the three code arrangements. *Application_A* is represented by the SPNs in Figures 4.21a and 4.21b. *Application_B* is represented in Figures 4.21c and 4.21d. Finally, *Application_C* is represented in Figures 4.21e and 4.21f. *Application_A* is represented by models with method calls in a sequential chain fashion. The first and second method-calls are dependent, represented in the model by $FINISH\_1\_START\_2 \in (proc\_time1)^{\bullet}$, where $(proc\_time1)^{\bullet}$ is the set of output transition of *proc_time1*. The second method-call is data dependent on the third method-call, represented by $FINISH\_2\_START\_3 \in (proc\_time\_2)^{\bullet}$. Following the same idea, *Application_B* is modeled using data dependence representation.

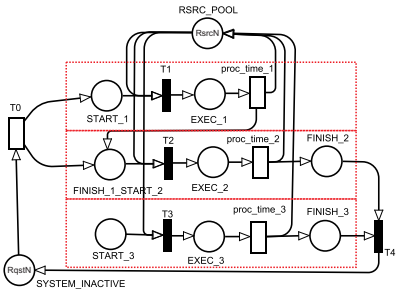
Application_B and *Application_C* are represented by SPNs comprising parallel tasks. Parallel tasks can be expressed by models including each individual task, a fork, and synchronization transitions. Two tasks are said to be parallel (or concurrent), if they are causally independent, enabling one transition firing either before or after another transition. Therefore, the model must encompass transitions such that its firing delivers tokens to more than one place.



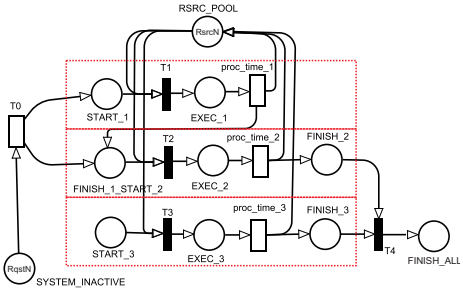
(a) SPN without Absorbing State Used to Calculate Throughput of *Application_A* (Three Sequential Method Calls).



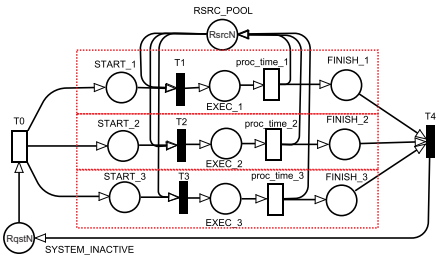
(b) SPN with Absorbing State Used to Calculate MTTE and CDF of *Application_A* (Three Sequential Method Calls).



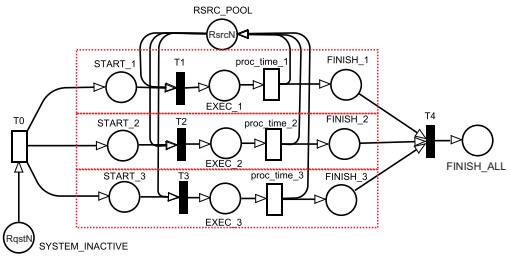
(c) SPN without Absorbing State Used to Calculate Throughput of *Application_B* (Two Sequential Method Calls and One in Parallel).



(d) SPN with Absorbing State Used to Calculate MTTE and CDF of *Application_B* (Two Sequential Method Calls and One in Parallel).



(e) SPN without Absorbing State Used to Calculate Throughput of *Application_C* (Three Parallel Method Calls).



(f) SPN with Absorbing State Used to Calculate MTTE and CDF of *Application_C* (Three Parallel Method Calls).

Figure 4.21: SPNs Generated by MCC-Adviser.

4.5.2 Model Validation

Many aspects may interfere in the similarity between the model results and the reality, such as connection with bad quality. To reduce the impact of errors (e.g., noise) in the measuring process, a statistical technique called bootstrap was adopted to validate the models proposed for Applications A, B and C (Efron and Tibshirani, 1993). Bootstrap is a resampling method: it obtains samples within a previously measured sample.

Table 4.2 shows that the throughput extracted from the models (SPN's Tp column) remains inside the respective confidence interval (CI). Therefore, this validation indicates that the generated models represents, statiscally proved, the reality.

Table 4.2: SPN Validation Using Bootstrap Technique.

	Throughput		Bootstrap	
Application	MCC-Adv.	Experiment	CI ($B\alpha/2$)	CI ($B[1-\alpha/2]$)
A	0.000642715	0.00064485	0.000629256	0.000661383
B	0.0008568	0.00086755	0.00083842	0.000894274
C	0.001057572	0.00102875	0.000985677	0.001070343

4.5.3 Model Solution

The proposed SPNs can be refined allowing one to obtain statistical information regarding the MCC environment. Hence, the applications were repeatedly executed under one VM of type *m1.medium* (1 CPU, 512MB of RAM, and a 10GB Disk), capturing the execution time for each method call. Only one specific 4MB picture was used as input. Next, a first SPN refinement was proposed by transforming the high-level transitions into exponentially distributed timed transitions by assigning the average delays to the respective transitions. Such transformation of transitions and delays assignment allows SPN to be solved and the throughput, MTTE, and CDF to be obtained.

Figure 4.22 presents the estimated throughput for applications A, B, and C. The number of considered resources ranged from one to four VMs because the approach takes into account higher granularity plus one. Since *Application_C* can be partitioned into three parts, four VMs were adopted as an upper limit. Thus, the totally sequential *Application_A* does not depend on the resource number. Therefore, the throughput remains constant considering different numbers of VMs. *Application_B* owns two blocks of independent code. Then, using two VMs the throughput increases. However *Application_B* has the same result for two, three, and four VMs since it cannot be partitioned into more than

two parts due to coupled code. Comparing the three applications, the throughput of *Application_C* is the highest because it considers the highest number of parallel tasks.

The MTTE for Applications A, B, and C can be viewed in Figure 4.23. Similar to the throughput metric, in *Application_A*, the MTTE does not vary when the number of VMs changes. However, Applications B and C have the benefit of parallelism, reaching saturation with two VMs for *Application_B* and three VMs for *Application_C*. Besides that, the software engineer may consider the MTTE desired by the user and compare with the results provided by MCC-Adviser. For example, if the final user demands a minimum MTTE around 2800ms for *Application_C*, the planner may adopt three VMs since two VMs do not allow reaching the maximal performance and four VMs do not offer improvement.

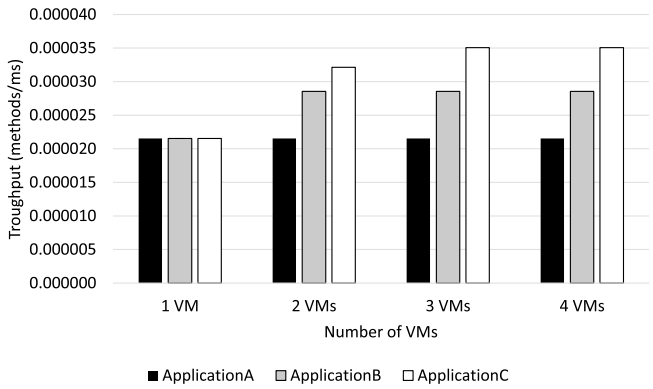


Figure 4.22: Throughput Evaluation Comparing Applications A, B and C.

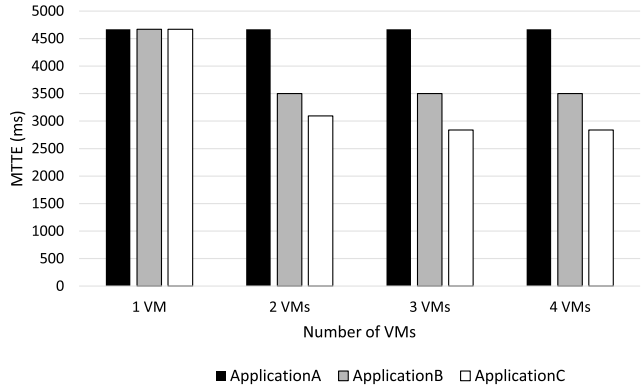


Figure 4.23: MTTE Evaluation Comparing Applications A, B and C.

Figures 4.24a, 4.24b, and 4.24c present CDFs that describe the execution time of each method. For each application, the *CDF* is plotted considering one, two, and three resources (VMs). The probabilities were computed from t = 0ms to t = 10,000ms.

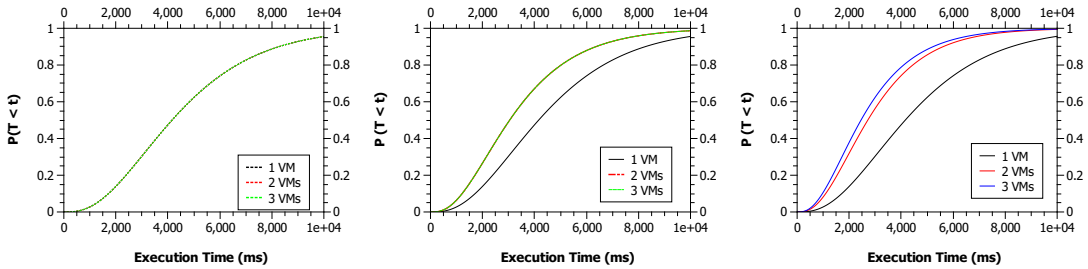
Although the applications have similar behavior, they are more likely to complete execution over time when the system is more decoupled. For *Application_A* (Figure 4.24a), the probabilities are the same for one, two, and three VMs. For *Application_B* (Figure 4.24b), the probabilities for two and three VMs are identical—the probability only differs when one VM is used. In this case, the probability of finishing the execution is smaller. *Application_C* (Figure 4.24c) has the highest probabilities for finishing execution faster. In addition, the probabilities for one, two, and three VMs are distinct.

Application developers and service providers willing to plan and design an MCC environment should be aware at when their applications are more likely to finish execution. The CDF may indicate such a moment through the maximum probability of absorption. Taking into account only one VM, the maximum probability of absorption is about 95% for the three applications. Best performance could be observed when using two or three VMs. *Application_A* does not reach 100% probability in any of the three scenarios. *Application_B* reaches 100% probability at exactly 10,000ms for two and three VMs. As previously mentioned, the probability for *Application_C* usually varies with the number of resources. However, for two and three VMs, 100% probability is achieved around 8500ms.

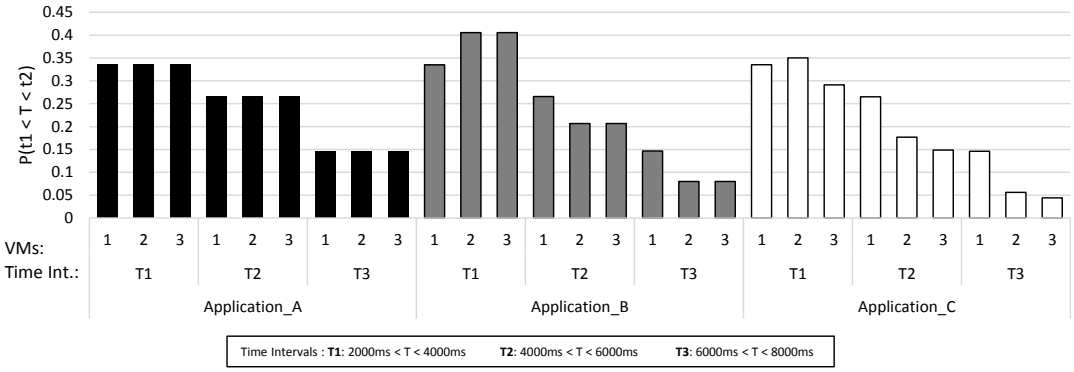
Willing to obtain the probability of absorption, the service provider may consider any time within the range. Final users may require that all applications finish by one specific time. Given that *Application_A* is the most constrained, the service provider should specify the observation of mainly *Application_A* in its Service Level Agreement. If the final user needing the application finishes execution by 5000ms, the probability for *Application_A* is always around 62%. Therefore, the service provider could agree to deliver the service by charging low prices due to infrastructure limitations.

Probability intervals can also be exploited using CDFs. Aiming to better analyze the applications, Figure 4.24d depicts the respective probabilities obeying three intervals. These intervals do not elucidate the cumulative probability starting from zero but rather the difference between two moments. Consequently, the probability is reduced as far as the interval values increase over time. Stated differently, the angle of the CDF line decreases over time. Hence, the more declined the line, the lower the probability. Observing the CDFs' cumulative probabilities, *Application_C* reaches higher results faster than *Application_B*. However, **Application_B** obtains higher values than **Application_C** when deploying two or three VMs. Due to such a myriad of interpretations, the application developer or service provider should also pay attention to probability intervals.

Moment matching (Desrochers *et al.*, 1995) could also be applied to obtain poly-



(a) Application_A **(b)** Application_B **(c)** Application_C



(d) Probability Intervals.

Figure 4.24: Probability Analysis of Applications A, B and C.

exponential distributions and the respective SPNs (Araujo *et al.*, 2011; Silvaa *et al.*, 2014; Costa *et al.*, 2015). By adopting moment matching, the planner may estimate what exponential-based probability distribution best fits the mean. Additionally, moment matching generates more accurate models, which can still be numerically evaluated. If non poly-exponential distributions are adopted, simulations should also be adopted. Figure 4.25 presents an SPN example of refinement using the moment matching technique. The SPN represents *Application_B* depicted in Figure 4.21d. The transition *proc_time_3* was refined as an example. A hypo-exponential distribution was attributed to such a transition. Therefore, a more accurate analysis could be done solving this emerging SPN.

4.6 Case Study Two - Time Metric - Face Recognition Application

In this section, a face recognition application is presented as a real case study aiming to enforce our statements with a nonliner code structure. Face recognition determines the match-likelihood of each face to a template element from a database. The widely

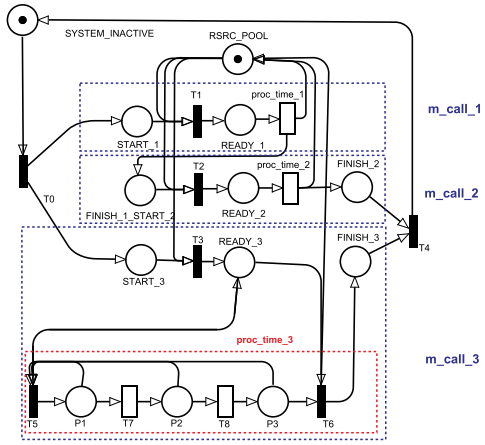


Figure 4.25: SPN Representing *Application_B* with a Hypo-Exponential Distribution.

accepted Eigenfaces approach was employed (Turk and Pentland, 1991). This process extracts the relevant information in a face image, encodes it, and compares the encoded face image with a database of models called face-space, similarly encoded.

Code 4.3 presents the analyzed class called *FaceRecognitionService*. The heaviest method, *recognize*, contains two heavy method calls. The first, *readFaceBundles*, constructs the face-spaces from a given directory. There must be at least 16 images in that directory and each image must have the same dimensions. The second method call, *checkAgainst*, performs the comparison between one photo and the face-space. This method call identifies the name of the most similar photo from the face-space and a Euclidean distance in that face.

```
1  public class FaceRecognitionService{
2
3     public static RecognitionResult recognize(String dirWithTrainedFaces, String
      photoToRecognize) {
4       EigenFaceCreator creator = new EigenFaceCreator();
5     creator.readFaceBundles(dirWithTrainedFaces); //m_call_1
6     String result = creator.checkAgainst(photoToRecognize); //m_call_2
7     String strresult = "Most closly reseambling: " +result+
8       " with "+creator.DISTANCE+" distance";
9
10    return new RecognitionResult(strresult);
11  }
12
13    public void readFaceBundles(String n) {
14    for (i = 0; i < bundles.length; i++) {
15      // Read each set of 16 images.
16        readBundle(filenames, set, i);
17      //m_call_1_1, m_call_1_2, m_call_1_3 ...
18      }
19  }
```

```
20 }
```

Code 4.3: Face Recognition Application Source Code.

The method calls *readFaceBundles* and *checkAgainst* are data dependent. However, with this application we intend to enhance our proposal by analyzing method calls repeatedly called. Therefore, *readFaceBundles* provides a loop that reads groups of 16 images at each iteration. All such processing can be performed in parallel managing replicated database.

The SPN model in Figure 4.26 represents the face recognition application with absorbing state. We reference *readFaceBundles* as *m_call_1*, *checkAgainst* as *m_call_2* and the “sub method calls" of *readFaceBundles* as *m_call_1_n* (where n ranges from 1 to 1000). Hence, the models are simplified, but in reality, the SPNs encompass 10,000 method-calls.

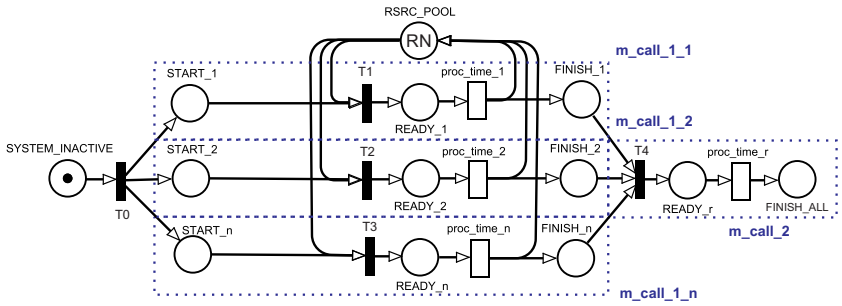


Figure 4.26: SPN Representing the Face Recognition Application with Absorbing State.

As a first step in the evaluation process we need to collect the delays for each method call using one resource. Thus, we have executed the application using a database of 16,000 photos in one VM (1000 iterations), registering the average execution times for the method calls, individually. We have repeated this process 30 times. The result was 0.0142125ms for *readFaceBundles* (in average) and 2.7ms to *checkAgainst*. Using these measurements, MCC-Adviser evaluated the use of 100, 500 and 1000 VMs and generated a CDF, depicted in Figure 4.27.

The probabilities were computed from t = 0s to t = 1.3s. According to Figure 4.27, the distances between the probabilities regarding 100 and 500 VMs are larger than the probabilities regarding 500 and 1000 VMs. The probability of finishing execution with 100 VMs becomes 100% only after 1.25s, whereas for 500 and 1000 VMs, this happens around 0.95s. With regard to the interval 0.3s to 0.45s, the following probabilities are

obtained: 24.4% for 100 VMs, 26.2% for 500VMs, and 27.4% for 1000 VMs. The interval $0.65s < T < 0.85s$ results in higher probabilities for 100 VMs than 500 VMs or 1000 VMs: 7.7% for 100 VMs, 5.1% for 500 VMs and 5.1% for 1000 VMs. Given so many tasks, at some point the probabilities for 500 and 1000 VMs are very close to each other. Such similarity is due to the accumulated effort in dealing with so many tasks concurrently.

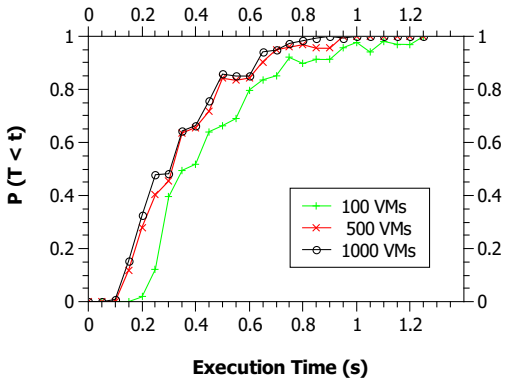


Figure 4.27: CDF of Face Recognition Application.

4.7 Case Study Three - Time Metric - GPU Study

Until now, MCC is limited to CPU code offloading. Inspired by the recent support for Graphic Processing Unit (GPU) computation on the cloud Ama (????), and the initial tentatives of using these GPU-capable virtual machines for data-intensive processing Shih *et al.* (2013), Pungila and Negru (2012), we envision a future where MCC will embrace the enormous possibilities offered by GPU computation offloading. General purpose computing on GPU (GPGPU) enables the possibility of optimizing the execution time of many parallel applications thanks to their large number of cores compared to the CPU. Imagine a normal smartphone being able to run the latest GPU-powered photo editor or to perform GPU-accelerated virus scanning Pungila and Negru (2012); all thanks to the cloud. We believe researchers will extend on previous works and integrate GPU code offloading into their offloading frameworks.

Unfortunately, aside from Amazon Ama (????), no other public cloud provider accommodates virtual machines with GPU support. Not only that, the only choices offered by Amazon are the *g2.2xlarge* and *g2.8xlarge* instances, both of them very powerful and expensive: $0.65/h and $2.6/h, respectively. The first instance type has 1

GPU, while the second one has four. The model is the same for both: High-performance NVIDIA, with 1,536 CUDA cores and 4GB of video memory, which is one the best graphic cards on the market. Final users would be enforced by the limited choice to pay for these very high-performance instances, even if their requirements were not so high.

We believe that in the near future this will not be the case anymore. The other providers are going to catch up Amazon and provide GPU VM instances as well. Not only that, we also believe that all providers will offer a broader spectrum of instance types so that users with different needs can choose accordingly and minimize their costs. In this thesis, we tackle the problem of choosing the optimal GPU instance in order to satisfy user's Quality of Service (QoS) requirements, while reducing his costs.

To make the presentation clear, we now consider a trivial example where the offloading framework should choose among three different GPU virtual instances. This example, presented in Figure 4.28, shows the execution time of a hypothetical application on the three GPUs. We assume GPU_1 is the worst performing, while GPU_2 and GPU_3 have almost the same performance. Since GPU_3 performs slightly better than GPU_2 it can have a higher cost. From the user's point of view however, this slight performance can be insignificant, so she can prefer saving money and use GPU_2.

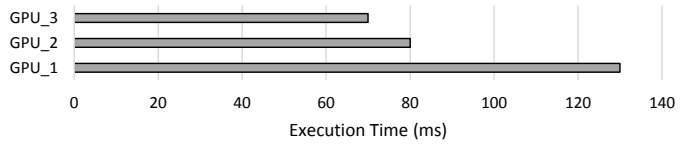


Figure 4.28: Execution time of a hypothetical application on three different types of GPU.

More concretely, consider the case where the developer has a requirement on the task execution time to be smaller than 120ms. With high probability, GPU_1 will not be able to satisfy this requirement. Choosing between GPU_2 and GPU_3 is the only alternative. Since GPU_2 has lower costs and can deliver the task result in time, the offloading framework should prefer that one instead of GPU_3.

Using the MCC-Adviser tool, mobile cloud offloading frameworks can automatically decide which instance type to use so that user's quality of service requirements are satisfied, while minimizing the costs. We have run a virus scanning benchmark application on a *g2.2xlarge* Amazon instance and measured the execution time, which was 5.48ms on average over 100 runs. We then divided the execution time by the number of CUDA cores in order to obtain the execution time per core (*EtpC*) and feed it to the MCC-Adviser tool. In absence of Amazon GPU instances with less CUDA cores, we defined four other

types with 98, 256, 512, and 1024 cores, assuming they use the same GPU model as the real one. Then, we used the previous measured execution time per core to estimate the probabilities for each of the defined instances.

In Figure 4.29 we present the results of the MCC-Adviser tool for the real GPU instance with 1536 cores and for the other instances defined by us. If the user requires her task to finish before 4.5ms, with probability higher than 95% even the less powerful instance would satisfy her needs. If the desired execution time was less or equal than 2.5ms, the probability of the less powerful instance drops to 76% and maybe another instance is better in this case.

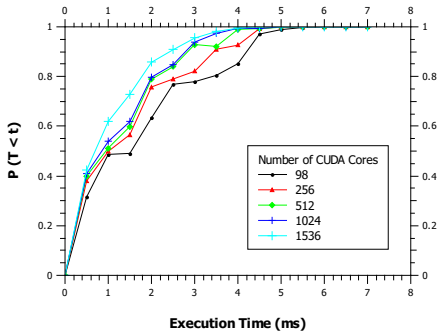


Figure 4.29: *CDF* line plot considering parameters from Amazon EC2 instance.

We are aware that the example reported has very small delay, however, considering real-time applications such result is totally possible. Shi et al. Shi *et al.* (2011), for example, proposes a real-time video encoding method for mobile cloud gaming in which some procedures take around 4ms in average. The purpose of this real case study is to simulate, for the first time, the choice of a GPU powered virtual machine in the cloud considering user's quality of service requirements. Our tool is extremely flexible. Producing the same results for other applications is just a matter of measuring the execution time per core and feeding the value to the SPN simulator.

Aiming to better present the results of this case study, a dynamic public web page is delivered through the URL: *http://cin.ufpe.br/∼faps/mcc-adv-gpu/*. The web page presents the CDF chart generated for this case study in conjunction with a way to access the specific probabilities. Thus, it is possible to visualize both, the probability of finishing execution before a give time $P(T < t)$, and the probability interval $P(t_1 < T < t_2)$.

4.8 Case Study Four - Energy Metric - Reduce Color Application

This section presents a case study observing the energy metric using an application for reducing images color.

4.8.1 Model Presentation

Aiming to evaluate the presented approach regarding energy metric we have used one of the three reduce color image processing application (the “ApplicationC"). The respective model is presented (again) in Figure 4.30:

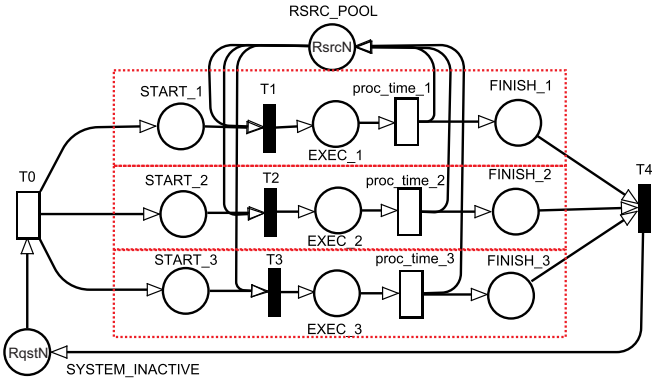


Figure 4.30: SPN of *Application_C* (Three Parallel Method-Calls).

The code structure with three parallel method-calls was chosen intending to highlight the energy consumption tendency as far as the parallelism increases.

4.8.2 Model Validation

Aiming to validate the energy model, we have used the *MCC-Logger-Client* for profiling the power and calculating how much energy the application spent. Three scenarios were tested: WiFi Cloudlet, WiFi Public Cloud and 3G Public Cloud. Each scenario was executed and monitored 30 times, collecting the mean values. The results have followed a normal distribution and then we generated 1000 values, extracting the confidence interval from that (CI) (Efron and Tibshirani, 1993; Silva *et al.*, 2014b). Table 4.3 presents the results comparison (from both: calculated by MCC-Adviser and experiment). The results shows that the energy extracted from the models (Model column) remains inside the

respective confidence interval. Therefore, this experiment provides evidence that our proposed SPN modeling with energy metric is reliable.

Table 4.3: SPN Model Validation

Scenario	MCETE (Model)	MCETE (Experiment)	Bootstrap - CI ($B\alpha/2$)	Bootstrap - CI ($B[1-\alpha/2]$)
WiFi Cloudlet	1037.24364	1038.6984	1036.74625	1039.95812
WiFi Public Cloud	4752.252667	4753.7064	4751.364983	4754.651928
3G Public Cloud	2585201.126	2585367.603	2584100.871	2587251.900

4.8.3 Model Solution

The same infrastructure presented at the previous section were used and three scenarios were considered. The first, the mobile device have offloaded tasks to a nearby Cloudlet using WiFi connection. In the second, the mobile device have offloaded to a public cloud (Amazon EC2) using WiFi connection. Third, the offloading was performed through a 3G connection to a public cloud (Amazon EC2).

The results for respective MCETEs — in logarithmic scale — are presented in Figure 4.31. Due to the maximum level of parallelism, three VMs were used as threshold. The experiment was conducted by using the same testbed presented in previous section and repeating the offloading tasks 30 times. The collected input parameters for each method-call were: the number of transmitted bytes and the communication time. The Figure 4.31 evidences that the 3G connection spends much more energy then WiFi. Observing WiFi isolated, it is necessary more energy in public cloud offloading then in private cloud offloading. However, in such scale it is not easy to observe the difference when increasing the number of VMs.

Figures 4.32a, 4.32b and 4.32c presents the isolated MCETE results for WiFi Cloudlet, WiFi Public Cloud and 3G Public Cloud, respectively. In the three scenarios, the energy consumption increases proportionaly to the number of VMs. Such an increment is due to the effort in dealing with multiple results. When offloading to one VM, only one result is received by the mobile device, but when offloading to N VMs, N results are received. In all three contexts, the difference between one and two VMs are higher than two and three VMs. We believe that close to the threshold of parallelism the difference between the number of VMs always decreases because it has no linear behavior. In that case, after 3 VMs (4, 5, etc) the MCETE results will be constant.

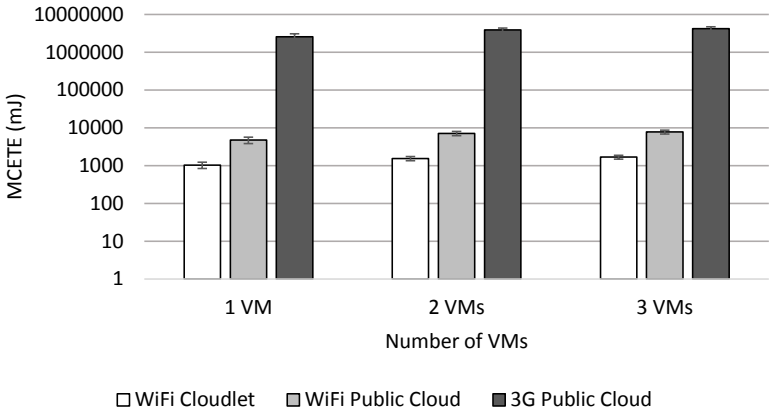
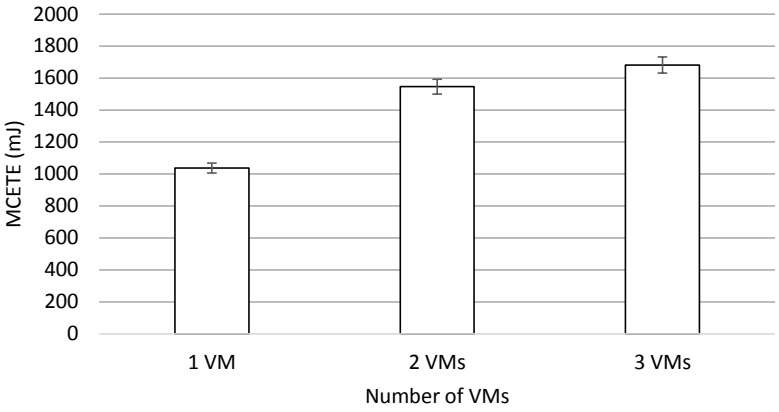
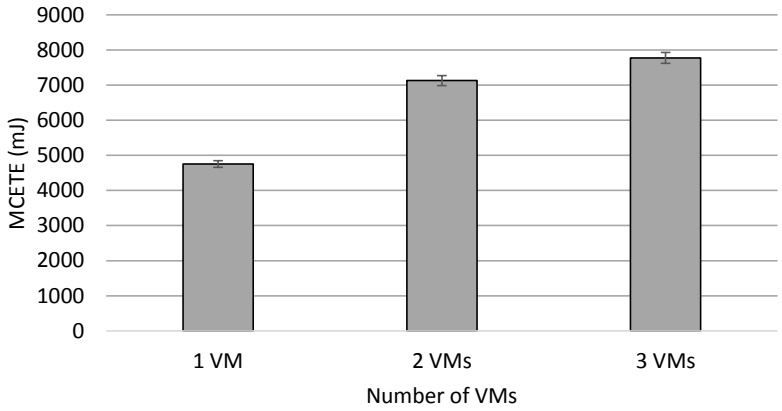


Figure 4.31: MCETE Comparison in Logarithmic Scale.

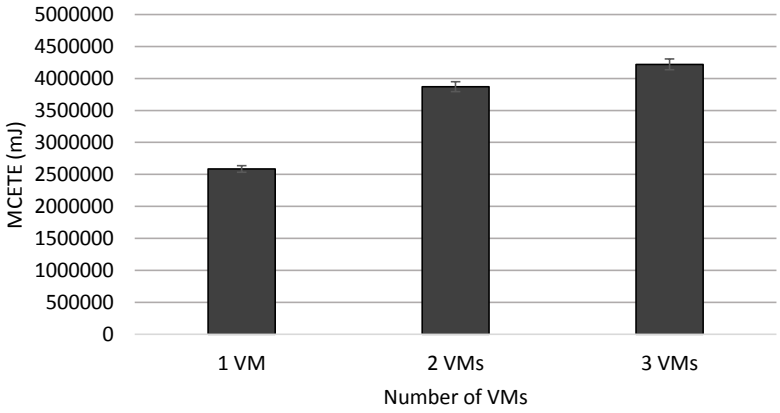
These results may be used for planning MCC infrastructures. MCC-Adviser guides the software engineer for deciding to change the code structure. For example, she may instead of distributing for n VMs to distribute for $X \times n$ VMs. The energy consumption increases but the total execution time tends to decrease as far as the number of VMs increases. Therefore, the software engineer should balance both metrics, execution time and energy in their final decision.



(a) WiFi Cloudlet



(b) WiFi Public Cloud



(c) 3G Public Cloud

Figure 4.32: MCETE for WiFi and 3G

5

Improving MCC Offloading Process

This section presents a smart allocation strategy to perform tasks offloading for mobile applications. The approach have considered an innovative balanced infrastructure parameters strategy.

5.1 Proposal Overview

Figure 5.1 presents a work-flow describing this part of the contribution. The objective here is to transfer the processing from mobile device to the cloud. The main activity is the offloading itself but there are other activities, needed for achieving the proposed goal. Following, the activities are detailed.

1. **Analyze the Application** - Not all applications are suitable for using offloading. Only those applications that demand a high level of processing may benefit from such a process. Another aspect is related to dependence of specific resources (GPS for example). Part of the application may not be offloaded since, at the cloud side, the tasks will not have all required information. Therefore, it is necessary to cautiously analyze the application and decide for applying offloading.

2. **Performance Evaluation (Before and After)** - There are many factors that may influence the offloading process, such as network connection and cloud unavailability. It is important to ensure that the application becomes more efficient by using the MCC approach. Therefore, a performance evaluation is needed before and after the offloading execution. The performance evaluation generates an evaluation report, summarizing results.

3. **Define the Partitioning Granularity** - According to (Liu *et al.*, 2015), partition

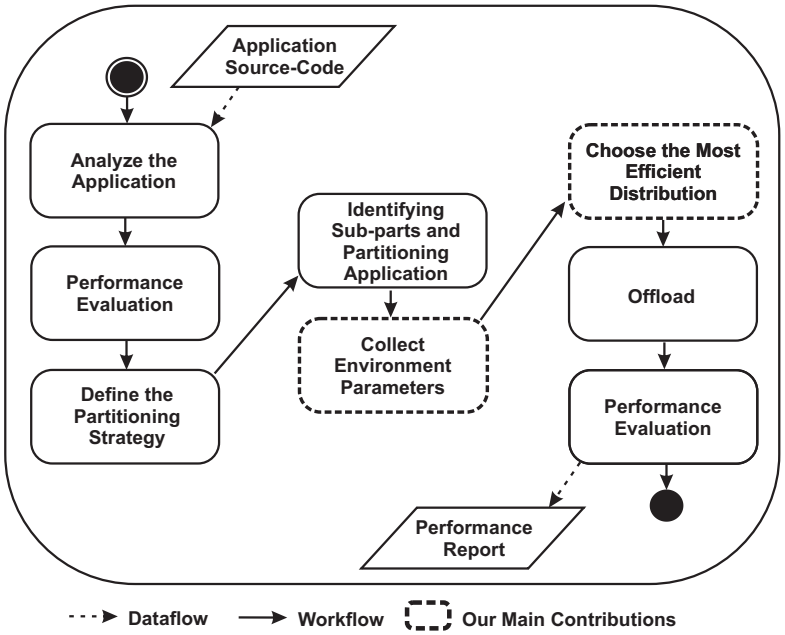


Figure 5.1: MCC Offloading - An Overview

granularity refers to the portion of the application which represents one atomic unit. One application can be offloaded without even any partition, in this case for example, the atomic unit is the application as a whole. Some other partitioning granularities are class, method, component, etc. In this work, the application is partitioned at method-level, but is important to note that during the offloading execution both are transferred: method instructions and input data. The performance results are directly dependent from the workload. In this work, image files are used as input data. Algorithms for image processing require much processing.

4. **Identifying Sub-parts and Partitioning Application** - Identifying the sub-parts of an application depends on the partitioning granularity. The sub-parts in this work are method-calls. This research have not focused on dynamic partition identification. The method-calls are manually identified and partitioned. Using the example of face recognition, there is a loop at the source code that performs the recognition for each human face in a photo. We have instrumented the code inserting offloading instructions.

5. **Collect Environment Parameters** - Aiming to offload the sub-parts of the application, the MCC environment must be monitored. MCC environment refers to the current state of the servers that process the offloading requests and the distance between mobile device and the cloud. More specifically, two parameters are col-

lected, RTT and current CPU consumption level. This part is considered a relevant contribution because previous work have not adopted both metrics together. Case studies following this strategy has shown a better application performance.

6. **Choose the Most Efficient Distribution** - This work proposes an algorithm for tasks distribution based on weighted parameters. The parameters (RTT and CPU) receives a weigh, meaning that one is “more important" then the other for an specific application. The objective of this research part was not to find the most optimized weight combination, but proving that the weighs may influence the offloading performance. This is the first time weighted parameters strategy is used in MCC, for the best of our knowledge.

7. **Offload** - Offloading includes to send, process and return the results to the mobile device. In this work all servers are configured to receive offloading requests. Each server has a copy of the application responsible for executing those offloaded method-calls.

5.2 An Smart MCC Offloading Process

Although the hardware of mobile devices evolve rapidly, they will continue being more resource poor than non-mobile hardware. Such fact occurs because, in the users point of view, the size, weight, and battery life have higher priorities than enhancing computational power. This is not just a temporary limitation of current technology but is intrinsic to mobility (Satyanarayanan *et al.*, 2009). MCC offloading in many situations can improve mobile devices performance. In terms of computing target, the ideal scenario was proposed by Satyanarayanan *et al.* (2009), with the concept of cloudlets. Cloudlets are nearby clouds (e.g: private cloud at an university) accessed through a high speed WiFi connection. Cloudlets are resource constrained, then the workload should be wisely partitioned and distributed.

The Round Robin algorithm is often used for that, as a simple-yet-effective method of distributing requests to a single-point-of-entry to multiple servers in the background. There is still a more sophisticated algorithm called Weighted Round Robin (WRR) (Nagle, 1988). In a Weighted Round Robin algorithm the “weight" determines the cost for processing at each server. The cost determines how many more (or fewer) requests are sent for each server; compared to the other servers on the pool. Round Robin is one of the most known algorithms for tasks distribution in MCC (Chen, 2015; Kosta and Aucinas,

2012; Chun *et al.*, 2011a). There are also some studies that have applied Weighted Round Robin with better results compared with the traditional one (Lin *et al.*, 2015; Abolfazli *et al.*, 2015). Although WRR is considered more accurate then RR, the WRR can be evolved.

This work proposes to evolve the Weighted Round Robin in MCC by assigning weights for the metrics used to calculate the offloading costs. The approach is called Smart Weighted Round Robin or just SmartRank. Figure 5.2 provides an overview of the proposed architecture. The idea is to partition the application and offloading it to the cloudlet considering that the target resources (Virtual Machines) have distinct configurations and states. Two parameters are initially adopted and balanced afterwards: the current CPU load and latency (Round-Trip Time). The result is a ranking with the respective offloading costs.

Regarding the cloudlet architecture, some decisions were necessary to guarantee that the SmartRank algorithm could be implemented. Aiming to avoid mobile device intrusiveness we propose the existence of an intermediary machine. This machine, named the Cloudlet Manager, deals with the algorithm for cost calculation and possible needed additional computing.

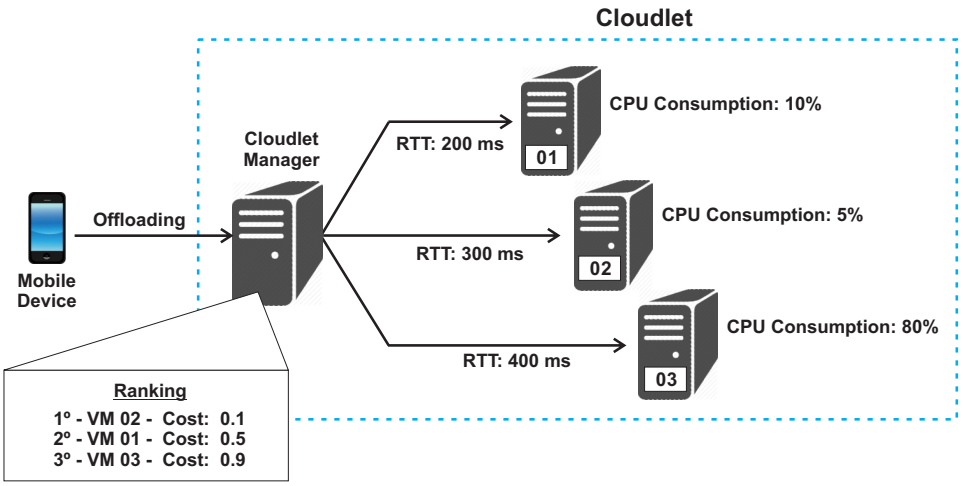


Figure 5.2: Virtual Machines Ranking - An Overview

In this work the number of method calls is directly proportional to the number of items that compose the workload. As aforementioned, the proposed algorithm takes into account two metrics, described following:

- **CPU utilization (*U*)**: Measures the current machine's CPU utilization percentage at a specific period. In other words, if the VM has two or more CPU cores then

the CPU utilization will be the average use of these cores. The higher the CPU utilization the lower the chances of allocating more requests to that particular VM.

- **Round-Trip Time (*RTT*)** (Dey *et al.*, 2013): Returns the sum of two "sub-metrics": the processing time on server-side and the Communication Time measured for only one face recognition. Similarly to the other metric, the higher the RTT the less requests the VM receives.
  - **Processing Time (*PT*)**: Represents the average time each VM type (large and small) takes to perform the recognition process in average. To get this measure we have run an experiment.
  - **Communication Time (*TT*)**: This value is obtained dynamically for each execution and represents the total time for a request to reach a VM and come back to the requester (cloudlet manager) without any processing.

Aiming to balance the metrics, each one is associated with a weight. The metrics U and RTT have the respective weights wU and $wRTT$, which summed up corresponds to the value 1.0. These weights are used to balance the formula that calculates the offloading cost (Ct) for each target machine. The measured metrics are normalized based on their amplitudes. The Min-max normalization method was adopted (Chakrabarti *et al.*, 2008). In this method, an attribute is normalized by scaling its values so that they fall within a small specified range: 0.0 to 1.0. Min-max normalization performs a linear transformation of the original data. Suppose that min_A and max_A are the minimum and maximum values of an attribute A. Min-max normalization maps a value v to v' in the new range $[min'_A, max'_A]$. The general corresponding formula for normalization is depicted in Equation $v' = \frac{v - min_A}{max_A - min_A}$ and the cost formula in Equation $Ct = ((wU \times NormU) + (wRTT \times NormRTT))$.

Figure 5.3 illustrates how the three macro components (Mobile Device, Cloudlet Manager and Cloudlet) interact with each other. The objective with Figure 5.3 is to highlight where the Smartrank algorithm interact with the offloading platform. For sake of clarity, as an example, lets consider the face detection/recognition mobile application used by many MCC research papers (Soyata *et al.*, 2012; Chen *et al.*, 2015; Jain *et al.*, 2011; Luo and Liu, 2010). There are seven steps from sending a photo to the Cloudlet and receiving the result.

1. **Sending Photo to the Cloudlet**: first the user takes a picture of a group of people and sends it to the cloudlet manager;

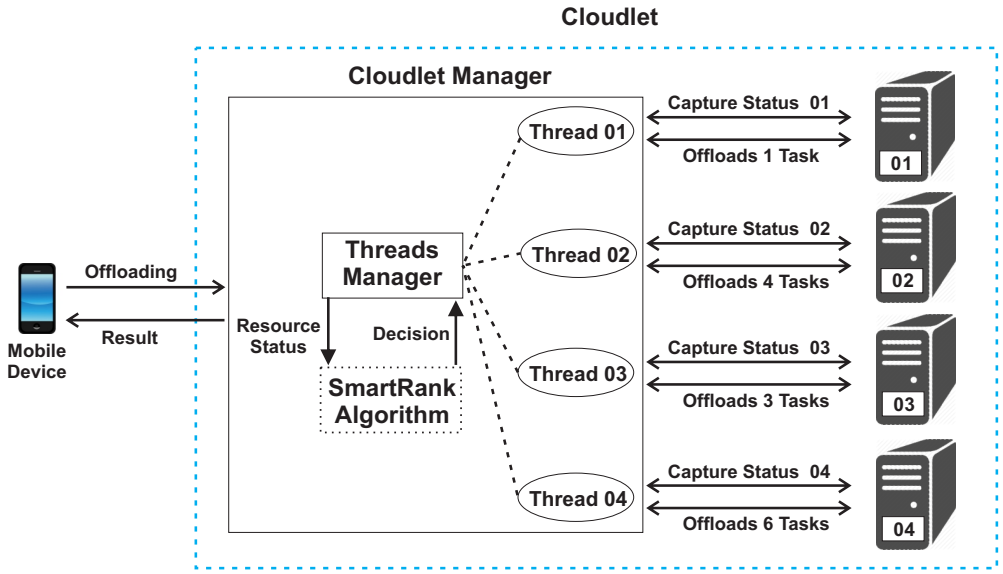


Figure 5.3: Offloading Steps Using Smart Ranking Approach.

2. **Input Partitioning (Faces Detection)**: the Threads Manager performs the faces detection and cropping;

3. **Computing Optimized Distribution**: the Threads Manager applies the SmartRank algorithm for calculating the best distribution of tasks (faces). In case of Round Robin Strategy such costs are equal;

4. **Creating Threads**: the Threads Manager sub-module creates a set of threads according with the number of available VMs and configure the number of tasks each one will transmit.

5. **Transmitting Tasks to VM Targets**: in this step the Cloudlet Manager effectively sends the packages of faces to the respective VMs;

6. **Face Recognition**: thereupon, the VMs execute the face recognition itself;

7. **Return Result to Cloudlet Manager**: when the recognition process has finished, the result is returned to Cloudlet Manager mobile device;

8. **Return Result to Device**: then, the result is returned to the mobile device.

The Equation $NT_{vm'} = round(\frac{1-Ct_{vm'}}{\sum_{N=1}^{K} 1-Ct_{vmN}} \times TNT)$ calculates how many tasks each VM will receive. Let the VM costs be $Ct_{vm1}, Ct_{vm2}, Ct_{vm3}, Ct_{vm4}, ..., Ct_{vmk}$. The Number of Tasks that a VM will process ($NT_{vm'}$) is obtained by the product of its cost percentage and

the Total Number of Tasks to process (*TNT*). We use $1 - Ct_{vm}$ because the lower the cost, the more tasks that particular VM will process. Therefore, the tasks are proportionally distributed among the VMs considering the $NT_{vm'}$ parameter. In other words, taking the face recognition as example, the machine with high cost will recognize less faces and that ones with lower cost will process more faces.

Based on these three Equations (5.2, 5.2 and 5.2) the SmartRank Algorithm was designed, and presented in Listing 1. Aiming to better present the algorithm, the code is divided into four procedures: *smartRanking* (the main procedure that calls the other auxiliary procedures), *calculateMinMax* (used to normalize the profiled metrics), *calculateIndividualCosts* (computes the costs per VMs), and *calculateNumberOfTasksPerVM* (finally it calculates the number of tasks each VM will receive). There is an entity in such a system that represents the VM and compose the variable *listOfVMs*. Such an entity could be expressed in many ways depending the programming language — as a class (in Java) or an struct (in C) for example. The important aspect is that a VM owns six atributes that are filed through these four procedures: *cost*, *normU*, *normRTT*, *currentU*, *currentRTT*, *nubmerOfTasks*. After executing *smartRanking*, every VM entity is configured with the corresponding number that they should offload. Next, the Cloudlet Manager just instantiates the threads and associates each VM entity.

Table 5.1 illustrates an example with real values extracted from an initial experiment, using 4 VMs and 14 faces. It is important to stress that the weights at the bottom of the table were just an example, and Section 5.4.2 shows what is the most effective balance for such weights.

Table 5.1: Example of costs calculation using 4 VMs and 14 faces.

Vm_Code	**U**	**PT**	**TT**	**RTT**	**Cost**	**NT_{vm}**
m1.m_a	69	21574,4	2345	23919,4	0,915	1
m1.m_b	12	21574,4	444	22018,4	0,330	4
m1.l_a	80	18551,0	182	18733	0,600	3
m1.l_b	2	18551,0	700	19251	0,040	6
Metric	**Wgt.**	**CostSum:**	2,11			
U	0,6	**TNF:**	14			
RTT	0,4					

Algorithm 1 SmartRank Algorithm

```
1: Global Variables:
2: wU ← 10          // For Example
3: wRTT ← 90        // For Example
4: TYPE_RTT ← "RTT"
5: TYPE_U ← "U"
6: maxRTT ← 0 minRTT ← 0 maxU ← 0 minU ← 0
7: procedure SMARTRANKING(listOfVMs, totalTasksNumber)
8:     CALCULATEMINMAX(listOfVMs, minRTT, maxRTT, TYPE_RTT)
9:     CALCULATEMINMAX(listOfVMs, minU, maxU, TYPE_U)
10:    CALCULATEINDIVIDUALCOSTS(listOfVMs)
11:    CALCULATENUMBEROFTASKSPERVM(listOfVMs, totalTasksNumber)
12: end procedure
13: procedure CALCULATEMINMAX(listOfVMs, min, max, metricType)
14:    auxValue ← 0 min ← Double.MAX_VALUE max ← Double.MIN_VALUE
15:    for vm ← each(listOfVMs) do
16:        if metricType == TYPE_RTT then
17:            auxValue ← vm.currentRTT
18:        else
19:            auxValue ← vm.currentU
20:        end if
21:        if auxValue < min then
22:            min ← auxValue
23:        end if
24:        if auxValue > max then
25:            max ← auxValue
26:        end if
27:    end for
28: end procedure
29: procedure CALCULATEINDIVIDUALCOSTS(listOfVMs)
30:    for vm ← each(listOfVMs) do
31:        vm.normRTT ← (vm.currentRTT − minRTT) ÷ (maxRTT − minRTT)
32:        vm.normU ← (vm.currentU − minU) ÷ (maxU − minU)
33:        vm.cost ← (wU × vm.normU) + (wRTT × vm.normRTT)
34:    end for
35: end procedure
36: procedure CALCULATENUMBEROFTASKSPERVM(listOfVMs)
37:    sumOfCosts ← 0
38:    for vm ← each(listOfVMs) do
39:        sumOfCosts ← sumOfCosts + (1 − vm.cost)
40:    end for
41:    for vm ← each(listOfVMs) do
42:        vm.numberOfTasks ← ((1 − vm.cost) ÷ sumOfCosts) + totalTasksNumber
43:    end for
44: end procedure
```

5.3 The SmartRank Prototype in Java

SmartRank algorithm (see Listing 1) was is implemented in Java language using the face recognition application as a benchmark. Intending to simplify the explanation we present a code version using pseudocode in Algorithm 5.1. The OpenCV (OpenCV, 2015) was used as an auxiliary library for processing the photos at the cloud side. OpenCV is an open source computer vision library with a strong focus on real-time applications. In our scenario, the OpenCV must be installed inside each VM and the databases' images must be replicated among them. As our focus is not improving the actual face recognition algorithm, we have adopted the wrapper JavaCV (JavaCV, 2015) to access the OpenCV, due to its expressive number of adapters.

The communication between mobile devices and the cloudlet-manager is implemented using sockets. We have chosen synchronous strategy because we judge real-time communication as more important than any other requirements. For the same reason, the messages are exchanged between cloudlet-manager and cloudlets with a synchronous remote procedure call (RPC) channel. There are many attractive aspects of RPC. One is clean and simple semantics: these should make it easier to build distributed computations, and to get them right. Another is efficiency: procedure calls are simple enough for the communication to be quite fast. A third is generality: in single machine computations, procedures are often the most important mechanism for communication between parts of the algorithm (Birrell and Nelson, 1984).

The SmartRank prototype have three components: SmartRank-Client, SmartRank-Cloudlet-Manager and SmartRank-Server. The SmartRank-Client is a Android application and the other two are traditional Java projects. The tool may be downloaded accessing the web-page containing all necessary information to install it: `http://cin.ufpe.br/~faps/smartrank`

5.4 Case Studies

The partitioning granularity, at first, could be any of those presented at the Background Section (2.2.1), because the algorithm simply considers a bunch of tasks. However, in MCC, Method Call Partitioning is one of the most intuitive approaches, because mobile applications are inherently organized in methods (Chun *et al.*, 2011b). Besides, method calls can bring high granularity as long as the methods are uncoupled. There are cases in which the computing processing concentrates under an unique method repeatedly

executed. One example is the mobile application for face recognition. As presented in Section 2.4, face recognition is the most adopted application as benchmark in MCC. The number possible parallel tasks will depend on the number of faces to be recognized. Listing 5.1 presents the respective source code in Java.

```
1  public class FaceRecognitionService{
2    public List<RecognitionResult> recognizeFaces(String dirWithTrainedFaces, String
       photoToRecognize){
3      List<RecognitionResult> results = new ArrayList<RecognitionResult>();
4      List<String> pathForFaces = DetectionService.detectAll(photoToRecognize);
5
6      for (String pathForFace: pathForFaces) {
7        results.add(recognizeOneFace(dirWithTrainedFaces, pathForFace));
8      }
9      return results;
10   }
11
12     public static RecognitionResult recognizeOneFace(String dirWithTrainedFaces, String
        pathForFace) {
13     EigenFaceCreator creator = new EigenFaceCreator();
14     creator.readFaceBundles(dirWithTrainedFaces);
15     String result = creator.checkAgainst(pathForFace);
16
17     String strResult = "Most closily reseambling: " +result+
18       " with "+creator.DISTANCE+" distance";
19
20     return new RecognitionResult(strResult);
21   }
22 }
```

Code 5.1: Face Recognition Application Source Code.

The method *recognizeFaces* receives a photo with a set of faces. Its first method call, *detectAll*, detects and separates the faces. Next, each face passes by the recognition process. Therefore, the method call *recognizeOneFace*, in the for loop, will be executed the number of detected faces. The parallelization can be done by executing the group of *recognizeOneFace* method calls in multiple resource targets.

In this section we present a proof of concept using the face recognition application. The mobile device is a thin client that sends a photo to the Cloudlet (with multiple faces) and receives the recognition result.

5.4.1 Case Study One: Local Execution

Parallel execution can be exploited much more efficiently on the cloud than on a smartphone, either using multiprocessor support or splitting the work among multiple VMs. Algorithms that deal with large amounts of data may benefit from parallel execution. Face

recognition, for instance, requires comparison of a particular face with a large database of pre-analyzed faces Kosta and Aucinas (2012). Since these types of applications handle a large data load, they are limited to completely run on a mobile device. In other words, not every workload is supported by standalone execution. In order to know what is the maximum load that a mobile device supports, it may be useful to guide new proposals of tools for partitioning and offloading. We performed a stress test to characterize the behaviour regarding the resource consumption of a recognition application running entirely on a mobile device to check its maximum power of execution. We tested the recognition of one face while using different databases containing 50, 100, 200, and 400 faces. At the end, we observed that the application could load and process the databases of 50, 100, and 200 faces but not the one of 400 faces, for as much as the application stops. We profiled memory, energy consumption and CPU in order for us to verify reasons for such a limitation.

Memory Profiling

The use of heap memory was profiled by instrumenting the application code (written in Java language). The library *java.lang.Runtime* was used, since it could provide information such as total memory and free space, which allowed us to obtain the percentage of memory used by application. Figure 3 illustrates that the memory traces were very similar among four database sizes (50, 100, 200, and 400). The most different behaviour was noticed on database size 400, inasmuch as application stopped working around the second 35 and memory did not reach a peak as had occurred to the other database sizes. The peak of memory had been reached when database was loaded into memory in order to perform the recognition. The peaks with databases 50, 100, and 200 are close one to each other (about 1% of difference). Therefore, for database 400, the peak memory consumption would likely be around 77% if the application had not stopped, which was still relatively far from 100% limit. Memory exhaustion was not the cause for the application stop.

Energy Profiling

Figure 5.5 shows the trace of energy consumption, measured with PowerTutor. The behavior of energy traces were also very similar among the database sizes 50, 100, and 200. The consumption level had a mild decrease as database size increased. This fact happened because Android devices are usually battery-powered; Android was designed to manage memory (RAM) to keep power consumption at a minimum. For example, when an Android app is no longer in use, the system will automatically suspend it in memory -

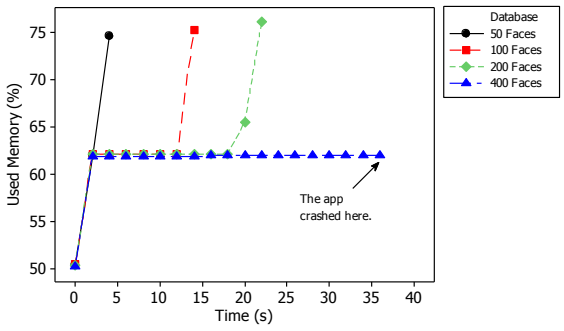


Figure 5.4: Memory Profiling.

while the app is still technically “open". Suspended apps consume no resources and sit idly in background until they are needed again. This has the double benefit of increasing general responsiveness of Android devices, since applications do not need to be closed and reopened from scratch each time, and also ensuring background applications do not consume power needlessly. Hence, Android limited the power consumption of face recognition app because memory usage kept constant until actual recognition point. The consumption fell immediately after recognition process was over. It was impossible to register consumption for the base of 400 faces, because PowerTutor application stopped working around the middle of the process. However, even at the databases loading—just before the app completes— the energy consumption had only a small increase.

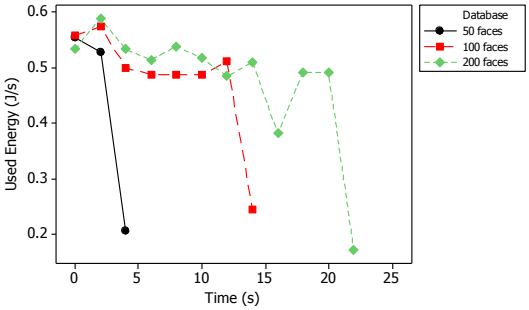


Figure 5.5: Energy Profiling.

CPU Profiling

The “proc" file system acts as an interface to internal data structures in the kernel of operating system. It can be used to obtain information about processes and other components of the system. The “/proc/stat" entry reads the Total CPU utilization of mobile phones when executing our recognition application. Figure 5.6 shows CPU utilization increases as the database size gets larger. Each execution obeys a similar

growth pattern, and afterwards, there is a drop of utilization just before application completion. This fall was not observed in the performance of 400 faces database because the application stopped when CPU reached the level of 98%. Once profiling showed memory did not get close maximum level, we conclude the crash occurred due to the CPU stress. This result indicates it is advisable to focus on optimizing CPU utilization rather than memory usage in such applications. This limitation in terms of hardware capacity motivates the use of cloud infrastructure to run this workload excess. In order to decide using or not cloud computing for this purpose, we can first evaluate the benefits of offloading it to a single server machine.

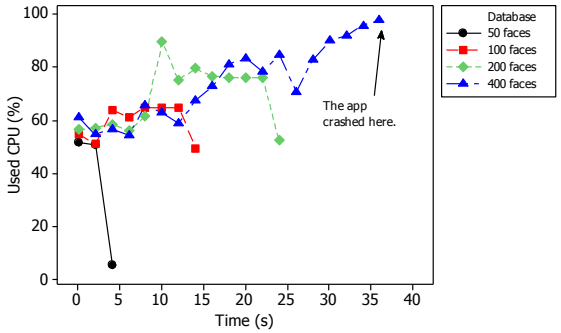


Figure 5.6: CPU Profiling.

5.4.2 Case Study Two: Round Robin Strategy

The round-robin experiment was assembled using Eucalyptus platform, comprising three physical machines. One machine was configured as the front-end, running the CLC, CC, SC, and Walrus. The two remaining ones ran the node controllers (NC). They executed the Linux CentOS 6 operating system and Eucalyptus platform 3.4.0.1. We used a 10/100 Mbps Ethernet network to connect the PCs through a single switch. The “m1.large" VM type was adopted with 2 CPUs (dual-core), 512 MB RAM, and 10 GB Hard Disk. As it is illustrated in Figure 5.3, we instantiated one, two, three, and four VMs. We used a database of 50 pictures, replicated among the VMs. The load balancer round-robin policy was employed to receive and redistribute one picture containing 4 equal faces. We repeated this experiment 30 times for one, two, three, and four VMs. For a closer view, the analysis is divided into two parts. First, the results for the offloading process are observed as a whole; and next, the individual steps are considered.

Analyzing the Offloading Process as a Whole

Figures 5.7 and 5.8 depict results for total energy consumption and total elapsed time, respectively. In general, results decreased for both energy and elapsed time as we increased the number of VMs. Comparing energy consumption using one and four VMs, there was a decrease of 44.69%, whereas elapsed time was reduced in 55.68%. This fact occurred because, by using round robin strategy, each face was recognized on a different machine. That was the preferable scenario to get the most from parallelism, since all four VMs have the same configuration.

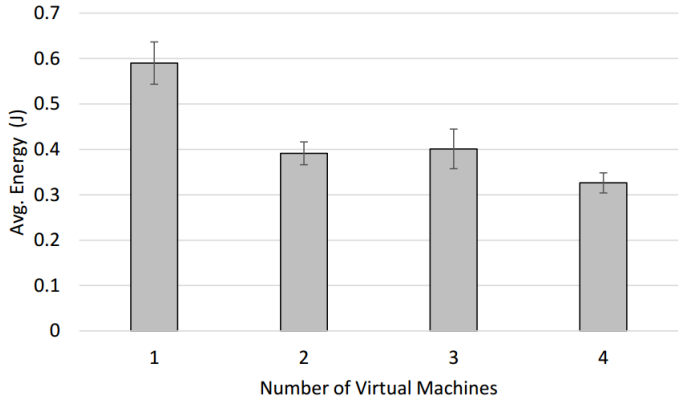


Figure 5.7: Energy saving through parallel remote execution.

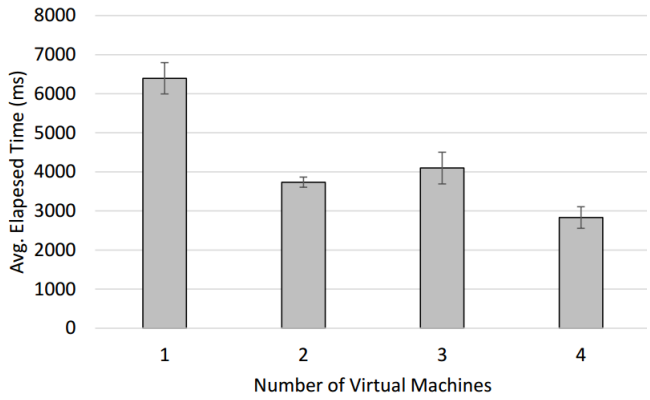


Figure 5.8: Elapsed time taken through parallel remote execution.

By observing the graphs it can be noted that the energy consumption and elapsed time were very similar considering two and three machines. Such similarity can be explained by the round robin faces distribution. As illustrated by Figure 5.9, with two machines, each of which received two faces. For three machines one received two faces and the other two machines received one. Since the execution was in parallel, the total “processing effort" would be approximately for that machine that took longer, it is, the server running the recognition of two faces.

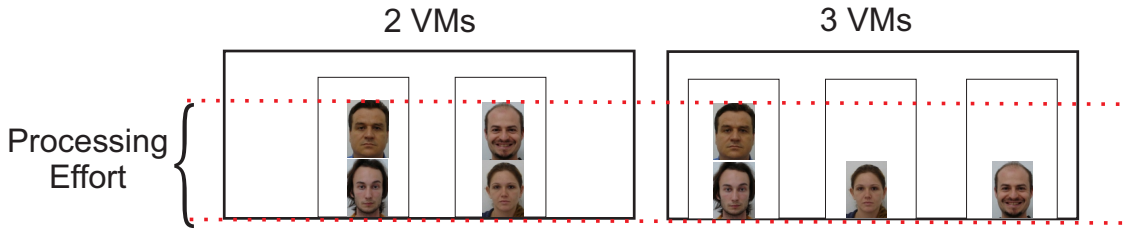


Figure 5.9: Offloading for 2 or 3 VMs.

.

Analyzing the Offloading Process by Steps

The offloading process is divided into four steps or types of efforts: detection, recognition, latency and others. Detection and recognition are considered primary actions. Latency and “others" are considered secondary actions. Latency represents the effort taken with communication between device and the target servers. “Others" are related to processing performed by the offloading agent. Such an offloading agent is the mechanism that creates and manages threads. These threads are responsible for sending and receiving offloading tasks. Figures 5.10 and 5.11 present the offloading process by steps. Figure 5.10 shows the proportion of average time for each step; and Figure 5.11 evidences the average time considering the distinct number of VMs.

Figure 5.10 evidences that the recognition step takes almost all the total elapsed time, about 80%. The recognition takes so long because each face is searched inside the database. The detection task (7%) does not takes long because the computation using Haar Classifiers Viola and Jones (2004) are very optimized. Latency (11%) is a critical constraint in MCC but in this context, using a nearby Cloudlet, the latency is not the botleneck. “Other" processing tasks are insignificant (2%) when compared to the previous actions.

Considering Figure 5.11, lets observe each step. In Figure 5.11a, the recognition time decreases as many VMs are included since the recognition is paralelized. In Figure 5.11b, the latency decreases as many VMs are included because the transmitted bytes are distributed among the VM targets. In Figure 5.11c, the time taken to manage threads increases as many VMs are included, however, increasing in a very low pace. Therefore, increasing for much many VMs will not impact much at the total elapsed time. In Figure 5.11d, by focusing on the scale the detection do not vary much (293∼298ms) because the processing effort is always the same: detecting 4 faces.

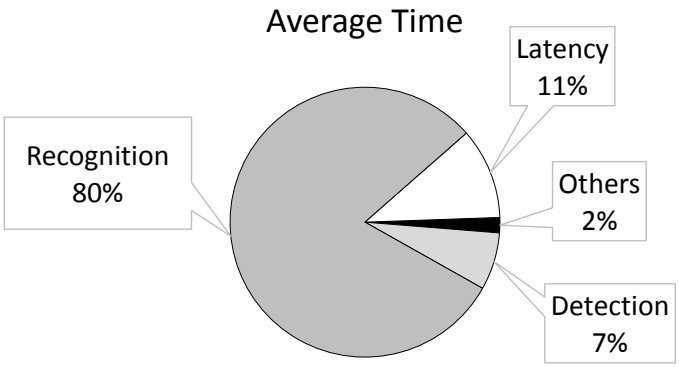
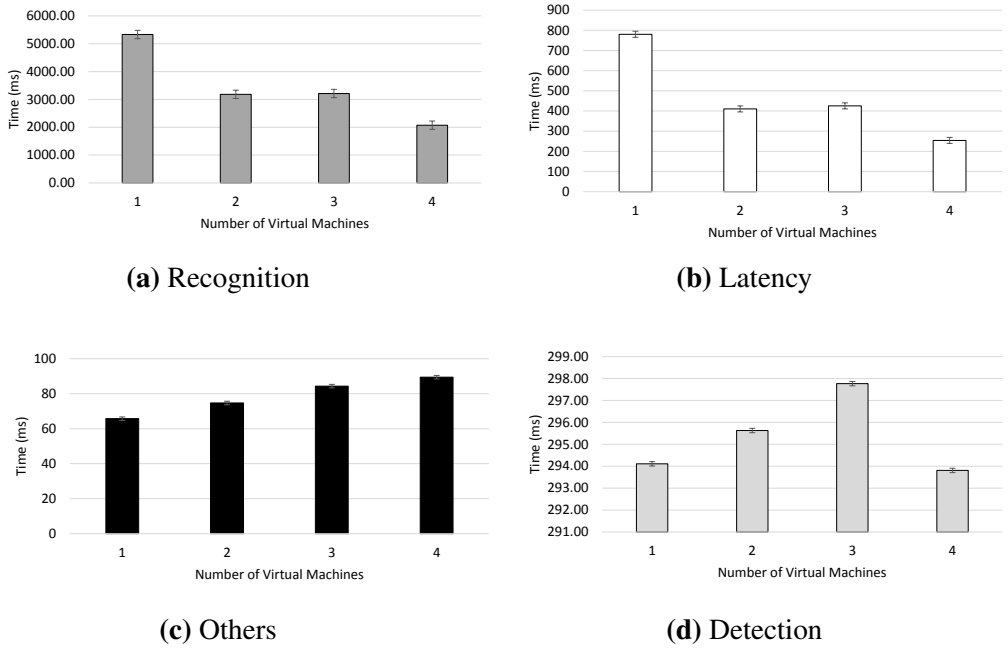


Figure 5.10: Average Time For Each Step at the Offloading Process.



(a) Recognition

(b) Latency

(c) Others

(d) Detection

Figure 5.11: Probability Analysis of Applications A, B and C.

5.4.3 Case Study Three: Smart WRR Strategy

Flores (Flores and Srirama, 2013) claims that the offloading is not a local decision process that happens just within the device, it involves a global understanding of the infrastructure. According to Tianyi *et. al* (Xing *et al.*, 2012), scheduling schemes for mobile cloud must consider multiple parameters such as computation and connectivity resources since the cloud environments are heterogeneous. We presented that the use of cloudlet federation decreased the mean response time around 48% considering the round-robin scheduling strategy. However, such strategy do not take into account the different VMs' capabilities

and latencies. For this reason something more sophisticated is needed. Thus, in this work we apply the Weighted Round-Robin (WRR) strategy that we call an smart strategy (Nagle, 1988).

This way, as aforementioned, SmartRank performs face detection and recognition through distribution of tasks among servers based on RTT and CPU utilization to make better use of heterogeneous infrastructures. Thus, we assign weights to these metrics because we suppose that depending on the scenario one metric can influence more the response time than the other. This aspect motivates in this context the following question: There is a calibration of weights that results in the lowest mean response time, executing in different scenarios in which the VMs have distinct initial CPU utilization levels? This section will present an study that aims to answer such a question.

The environment was assembled with one cloudlet comprising three machines with the same hardware configuration: Intel Core i7-3770 3.4 GHz CPU, 4 GB of RAM DDR3, 500 GB SATA HD. One machine is configured as the front-end, running the CLC, CC, SC, and Walrus. The remaining two run the node controllers (NC). They execute the Linux CentOS 6 operating system and Eucalyptus platform 3.4.0.1. We use a 10/100 Mbps Ethernet network to connect the PCs through a single switch.

Two VM types were adopted: m1.medium (1 CPU, 512MB Mem., and 10GB Disk) and m1.large (2 CPUs, 512MB Mem., and 10GB Disk). As depicted in Figure 5.3, we simulate two cloudlets with four VMs (2 m1.medium and 2 m1.large). To narrow the scope, the mobile device is only responsible for sending raw images to the cloudlet-manager and this process is not repeated during the experiments. Constant mobile transfer time is added in every experiment.

Before the calibration process we had to estimate the metric *processing time (PT)*, referring to the time for a VM to perform recognition without considering Communication Time. RTT is composed of Communication Time (TT) that is dynamically obtained and the processing time (PT) that is an estimation resulted from an experiment described in this section. Hence, in the case of TT metric, for each scheduling execution a simple request is spread for all VMs and then their response times are recorded. In the case of PT metric we shall not do the same because it could significantly decrease the scheduling performance, so such metric is obtained as a mean value through experiment.

We registered the PT mean for one VM instance of type *m1.medium* and other of type *m1.large*. They received a specific workload 80 times composed of one picture with 17 faces. The VM of type *m1.large* obtained a lower PT mean (18551 ms) than the *m1.medium* (21575 ms). This result was expected due to its different computational

power (*m1.medium* has 1 CPU core and *m1.large* has 2 CPU cores).

To use these PT mean values only makes sense if the two samples are statistically different, otherwise the type of VM would not influence the desired response time. So, aiming to ensure that these means could be used as a relevant metric for ranking, we again applied the t-test to verify whether these means were statistically different. Considering 95% of confidence, we assume the normality of both samples (p-values equal to 0.735 and 0.300). The t-test showed that there is a significant difference between them: VM type m1.medium (M=21575, SD=32.26) and VM type m1.large (M=18551, SD=27.10); T(153)=642.02, p=0.000. Figure 5.12 depicts a box-plot illustrating the samples distance.

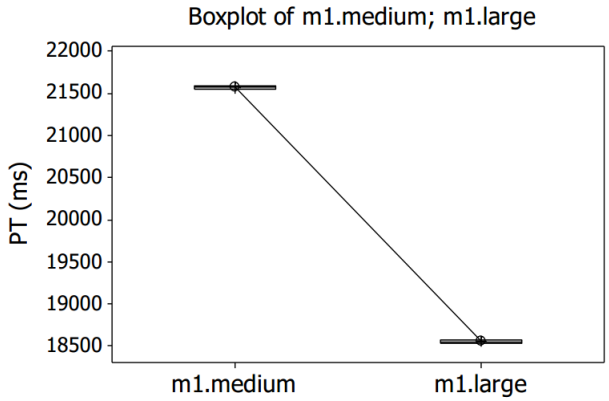


Figure 5.12: Box-Plot graph to illustrate the distance between the samples.

Algorithm Calibration

Aiming to find the calibration of weights that results in the lowest mean response time we arranged different scenarios where the initial CPU utilization of four VMs (2 m1.large and 2 m1.small) are different. We performed experiments using the “real" RTT as a dependent variable. It is important to stress that we refer two types of RTTs. The first is used by the scheduling algorithm, including the pre-calculated PT values and instant TT. The second is the “real" RTT, recorded in our experiments. Table 5.2 shows the factors and their respective levels. We have chosen the weight balance as a factor because we want to find the weight balance that results in the lowest RTT. The second factor is the initial CPU utilization level because depending on the level of this metric a VM should not receive more requests.

In the case of weight balance factor we have chosen three calibrations, considering RTT (acronym ’R’) and CPU utilization (acronym ’U’). First, giving more importance for U (with 20% for R and 80% for U). Second, giving more importance for R (with 80% for

Table 5.2: Factors and the parameters chosen as relevant.

Factors	Wgt. Balance (%)	Initial CPU Utl. (%)
Levels	20R80U	m.a:10,m.b:20,l.a:30,l.b:40
	80R20U	m.a:40,m.b:30,l.a:20,l.b:10
	50R50U	m.a:10,m.b:10,l.a:20,l.b:20

R and 20% for U). Third, considering them equally important (with 50% for R and 50% for U). We have tried others values, however the above 80 and below 20 the difference was inexpressive.

We have simulated the initial CPU Utl. level using the *LookBusy*[1] tool that generate fixed and pre-configured loads on CPUs. The acronym m.a means m1.medium.a, m.b means m1.medium.b, l.a means m1.
large.a and l.b means m1.large.b. The letters "a" and "b" are used only to identify the two VMs of each distinct type. We configured three scenarios setting arbitrary CPU utilization levels for the four VMs varying the load from 10 to 40 percent. The acronyms presented in Table 5.2 will be used it the remainder of the chapter.

As the experiments were executed on the same network, we did not consider Communication Time as a factor, setting a fixed value equals to 3, a value previously observed in the previous executions. To capture the real RTTs, we just instrumented the source code before and after the process in the cloudlet-manager, registering the difference in milliseconds in a text log. For each sequence of execution, the VMs were cleaned (processes stopped) and the text logs were also recreated.

We have adopted the statistical method factorial Design of Experiments (DOE) (Montgomery and Montgomery, 1984), as we have two factors to obtain the desired measures, and our intent is to study the impact of each factor on those measures to finally extract the best weights balance. Considering the two factors (weight balance and initial CPU utilization), and three levels for each one of them, there are nine experiments to run, which are described in Table 5.3, presenting the real RTT mean and respective standard deviations (SD). In order to get results in an acceptable confidence level, we decided to use a photo with 15 faces and run 35 replicas for each executions, yielding a total of 315 experiments.

The effect and relevance of each factor and their interactions were computed by using the results of the real RTT time, shown in Table 5.3 applying the method factorial Design of Experiments (DOE). Table 5.4 introduces the respective estimated effects.

The results in Table 5.4 showed that the factor with the greatest impact is weight

[1] hPT://www.devin.com/lookbusy/

Table 5.3: Results of each treatment of the experiment.

Initial CPU utilization	Wgt. B.	RTT M.	SD
m.a:10,m.b:10,l.a:20,l.b:20	20R80U	988.14	31.22
m.a:10,m.b:10,l.a:20,l.b:20	80R20U	1045.06	43.73
m.a:10,m.b:10,l.a:20,l.b:20	50R50U	1022.86	153.56
m.a:10,m.b:20,l.a:30,l.b:40	20R80U	1078.69	98.68
m.a:10,m.b:20,l.a:30,l.b:40	80R20U	1194.71	233.05
m.a:10,m.b:20,l.a:30,l.b:40	50R50U	1163.63	174.48
m.a:40,m.b:30,l.a:20,l.b:10	20R80U	1018.06	51.27
m.a:40,m.b:30,l.a:20,l.b:10	80R20U	1162.94	270.51
m.a:40,m.b:30,l.a:20,l.b:10	50R50U	1111.00	140.47

Table 5.4: Estimated effects and relevances for the RTT mean time.

Factor	Effect	T	Relev.	P
weight_balance	110.93	4.24	61%	0.000
initial_U	-65.73	-2.51	36%	0.013
weight_balance*initial_U	-5.1	-0.19	2%	0.846

balance (weight_balance), generating an effect with a relevance of 61% (p=0.000, as it is lower than 0.05, it is significant). It means that a variation in such a balancing may increase or decrease the resulting response time over the face recognition. The initial CPU utilization (initial_U) also influences the real RTT, but only by 36% (p=0.013). The interaction between both factors resulted in a p-value=0.846, indicating absence of mutual influence. Thus, we analyse the effect of factors on an individual level.

The Pareto chart (see Figure 5.13) depicts the importance of an effect by its absolute value, drawing a reference vertical line on the chart. The more the effect extends this line the more it influences the dependent variable (that is the RTT in our study). The line indicates the minimum magnitude of statistically significant effects, using the criterion of statistical significance $\alpha = 0.05$. Figure 5.13 presents a Pareto Chart in which can be observed the significant influence of weight factor (weight_balance) compared with the initial CPU utilization of VMs before sending faces for recognition (initial_U). The graph also shows the little interaction between the factors (term AB), without statistical significance.

The effect of one factor may depend on the level of the another factor, resulting in the so called factors interaction. This phenomenon may be evidenced when plotting each level of a factor and keeping the level of a second factor constant. Thus, it compares the relative strength of the effects across factors observing the existence of a pattern, if noted, this pattern means that there is no interaction. For such intent we use a bar plot

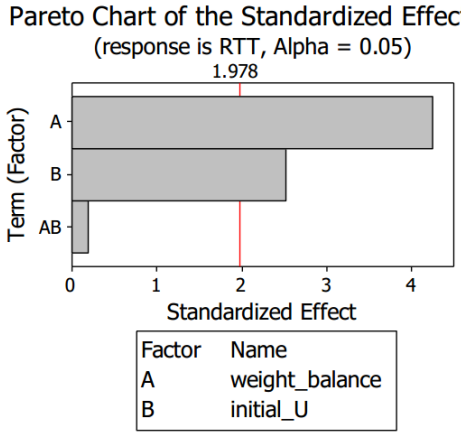


Figure 5.13: Pareto Chart representing the effects of each factor. The red line represents the minimum magnitude of statistically significant effects.

(Figure 5.14) looking for a pattern on the factor initial CPU utilization (initial_U) whereas keeping the weight balance (weight_balance) constant. By the figure we can reinforce that there is no interaction between the factors, since for each weight balance level the RTT increases proportionally to the initial CPU utilization.

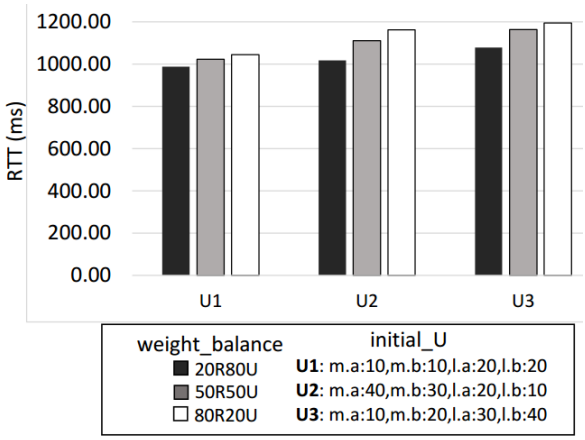


Figure 5.14: Bar plot with the level of relationship between the factors.

Since the interaction between the factors is not significant we can treat the factors individually. Figure 5.15 presents the average result for each one of the factors and then we can make some conclusions about the factor levels (from left to right side). First, as observed in the last plot shown in Figure 5.14, the different weight balances present distinct results. The best performance was when using the balance of 20% for RTT and 80% for CPU utilization metric, it obtained a 5.5% faster result comparing to the average of the three real RTTs. It can be explained by the invariance of the RTT, as the PT is a

fixed number and the Communication Time is equal for all VMs in our experiments. In the right side of the graph, the factor initial CPU utilization (initial_U) resulted in a higher real RTT mean when the values were in a high level, that is, when the VM owned an expressive initial workload. Thus, when the VMs were with this level of CPU utilization “m.a:10,m.b:20,l.a:30,l.b:40", the performance was the worst because the instance types “m1.large" were running more busy, with 30% and 40% of the CPU utilization.

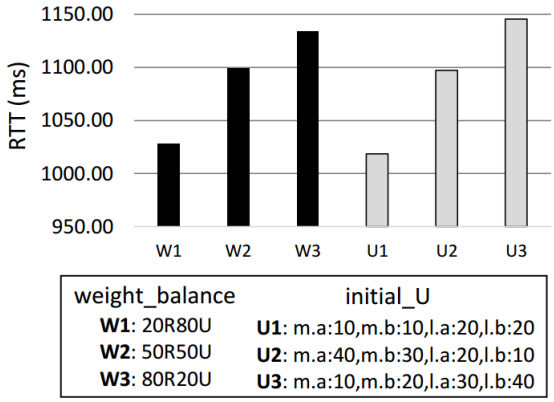


Figure 5.15: Bar plot showing the relative effects of each level.

Thus, we conclude that the weight balance “20R80U" will result in the best face recognition performance under some assumptions: assuming that all cloudlets are near from each other (similar Communication Time); the pool of resources is divided into two groups of VM types (medium and large); and the VMs’ CPU utilization do not exceed 40%. This scenario is plausible because our proposal aims to have a federation on cloudlets near the client and close to each other. The RTT can be decreased if more resources are included. SmartRank can redirect the requests every time the cloudlets federation is overloaded, however this is not covered in this research.

6

Conclusions and Future Work

This PhD research achieved a number of results in the areas that it has explored so far. The major contributions are the new insights of how to save energy and time of MCC applications. We believe that smart offloading allocation algorithms and system modeling have provided meaningful success, despite these mechanisms may still evolve and conquer further improvements.

First, this research introduced the results of a systematic mapping study about benchmark applications used in mobile cloud computing by investigating scientific literature production. Given the current state of MCC research, we judge that there are few studies with controlled experiments using real applications. We believe that this mapping study generated state-of-the-art information about the main issues because the studied subject can be understood through the provided answers. In future work, more systematic mappings should be conducted to acquire further experience to aid new experiments. This part of the thesis was published in one Symposium (Silva *et al.*, 2015a) and one journal (Silva *et al.*, 2016a).

This research introduces a modeling approach to represent code dependency of mobile applications using Stochastic Petri Nets (SPNs). The approach provides graphs depicting Throughput, Mean Time to Execute, Mean Consumed Energy to Execute, and Cumulative Distribution Functions (*CDFs*). A tool called MCC-Adviser was proposed and evaluated using a private cloud. Such a tool aims at assisting software engineers to plan their mobile cloud infrastructure with very little effort. To the best of our knowledge, this is the first work to use SPN in the field of MCC with automatic nature. One version of MCC-Adviser is delivered in a public web application through the URL: *http://cin.ufpe.br/∼faps/mcc-adv/*. This work can also be used in conjunction with other techniques to increase mobile cloud performance. This part of the thesis was published by one conference (Silva *et al.*, 2015b) and have two other journal papers in revision.

In 2013 Amazon launched the first virtual machines with GPU support. Two years later, we are still stuck with the same instances. No other public cloud company provides these type of machines yet, due to technological difficulties and high costs, in our opinion. Mobile devices have been broadly using cloud computing to increase applications performance. Following the successful trend of mobile application offloading towards powerful servers, we believe that in the very near future mobile GPU offloading will be a reality. This work presented an approach to represent GPU parallel processes with Stochastic Petri Nets. Using such a representation we implemented a tool, called MCC-Adviser, that can simulate GPU executions and plot cumulative distribution functions in a highly flexible way. Using the MCC-Adviser, we plot the probabilities of satisfying user requirements when using GPUs with different number of cores. We found that user can reduce her costs by opting for a less powerful GPU virtual machine, while still satisfying application's requirements in terms of execution time.

Other original contribution in this work is a partitioning and offloading technique that distributes tasks of mobile applications to the cloud. The approach intends to minimize response time of mobile applications by using cloud computing with heterogeneous communication latencies and compute power. We evaluated the smart approach by experimenting face detection and recognition algorithms on Android devices. To the best of our knowledge, this is the first work showing such a strategy, comprising private cloud and the weighted metrics approach. The proposed tool, SmartRank, integrates mobile devices (e.g: smartphones), the cloudlet manager, and cloudlets. This work focused on describing the strategy and a sensitivity analysis of SmartRank to find suitable parameters that would result in good results considering response time. The experiments evidenced that: the use of cloudlets federation is feasible for face recognition, since maximization of cloudlet capabilities improved the response time of recognition process by 48%, that is, instead of one resource, multiple machines can solve faster a recognition task; and it was possible to find a calibration for the metrics CPU utilization and RTT based on weights, a functionality not applied so far in our known literature. This part of the thesis was published by one conference (Silva *et al.*, 2015c) and two journals (Silva *et al.*, 2015d) (Silva *et al.*, 2016b).

Finally, this research has provided one more step in the maturation of MCC, but mobility will continue being a hard research challenge.

6.1 Future Work

Following, we list some possible future work:

- **Using Stochastic Models for Predicting MCC Costs**: This work has adopted only private clouds during the experiments. We envision the future of MCC taking advance of public clouds and modeling for considering financial aspects. This work presented a way of predicting how much time an application could spend by offloading tasks. Public clouds usually charge their customers based on how long time their VMs were used. Therefore, using these two information (price and predicted time) it could be possible to calculate the costs for public cloud offloading.

- **Energy Profiling at Source-Code Line Level**: The energy profiling mechanism proposed in this work was evolved from PowerTutor application. This application is not precise and more appropriate only for Google phones. Besides, today, there is no application capable of profiling energy spending at source-code line level. This possibility could enable programmers evaluating their new applications aiming at extending the mobile device autonomy.

- **Finding the Most Efficient Weigh Balance in SmartRank Strategy**: This research has shown that setting weighs for balanced metrics could increase performance during offloading execution. However, this work did not present the combination of weights that could achieve the optimal result. Artificial neural network could be applied aiming to solve that problem.

- **Extending the Stochastic Models with Other Metrics**: The SPN models representing method-calls could be extended to study dependability, including availability and reliability. Although the related work presented some studies exploring these metrics, their models did not observed the application at source code level.

- **Explore MCC Offloading with Wearable Devices** : Internet of Things is a hot topic today. Although in this work we mention mobile devices in general, in practice we only focused on smartphones. Wearable devices, such as smart swatches, are even more limited compared with mobile devices. Both approaches/tools of this work (SmartRank and MCC-Adviser) could be applied using applications that runs under these gadgets.

Bibliography

(????). Amazon web services - amazon ec2 instances. https://aws.amazon.com/en/ec2/instance-types/. Accessed: 2015-07-28.

(2015). Centos. https://www.centos.org/. Accessed: 2015-07-28.

Abolfazli, S., Gani, A., and Chen, M. (2015). Hmcc: A hybrid mobile cloud computing framework exploiting heterogeneous resources. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*, pages 157–162.

Abraham, A., Mauri, J. L., Buford, J., Suzuki, J., and Thampi, S. M. (2011). *Advances in Computing and Communications, Part IV: First International Conference, ACC 2011, Kochi, India, July 22-24, 2011. Proceedings, Part IV*. Springer Publishing Company, Incorporated, 1st edition.

Antonio, R. (2013). "who coined 'cloud computing'?". In *Technology Review. MIT.* Available on https://www.technologyreview.com/s/425970/who-coined-cloud-computing/.

Araujo, C., Maciel, P., Zimmermann, A., Andrade, E., Sousa, E., Callou, G., and Cunha, P. (2011). Performability modeling of electronic funds transfer systems. *Computing*, **91**(4), 315–334.

Araujo, C., Silva, F., Costa, I., Vaz, F., Kosta, S., and Maciel, P. (2016). Supporting availability evaluation in mcc-based mhealth planning. *Electronics Letters*, **52**(20), 1663–1665.

Araujo, J., Silva, B., Oliveira, D., and Maciel, P. (2014). Dependability evaluation of a mhealth system using a mobile cloud infrastructure. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 1348–1353. IEEE.

Ardito, L., Procaccianti, G., Torchiano, M., and Migliore, G. (2013). Profiling power consumption on mobile devices.

Balan, R. K. (2006). *Simplifying cyber foraging*. School of Computer Science, Carnegie Mellon University.

Balbo, G. and Chiola, G. (1989). Stochastic petri net simulation. In *Proceedings of the 21st Conference on Winter Simulation*, WSC '89, pages 266–276, New York, NY, USA. ACM.

Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, **2**(1), 39–59.

Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. S. (2006). *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons.

Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, page 6. Ieee.

Callou, G., Maciel, P., Andrade, E., Nogueira, B., and Tavares, E. (2008). Estimation of energy consumption and execution time in early phases of design lifecycle: an application to biomedical systems. *Electronics Letters*, **44**(23), 1343–1344.

Callou, G., Maciel, P., Tavares, E., Andrade, E., Nogueira, B., Araujo, C., and Cunha, P. (2011). Energy consumption and execution time estimation of embedded system applications. *Microprocessors and Microsystems*, **35**(4), 426–440.

Callou, G., Ferreira, J., Maciel, P., Tutsch, D., and Souza, R. (2014). An integrated modeling approach to evaluate and optimize data center sustainability, dependability and cost. *Energies*, **7**(1), 238–277.

Campos, Eliomar; Matos, R. M. P. C. I. S. F. and Silva, F. A. (2015a). Performance evaluation of virtual machines instantiation in a private cloud. In *Services (SERVICES), 2015 IEEE World Congress on*, pages 319–326.

Campos, Eliomar; Matos, R. M. P. S. F. and Silva, F. A. (2015b). Stochastic modeling of auto scaling mechanism in private clouds for supporting performance tunning. *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*.

CCS-Insight-Forecast (2015). Smartphone sales to peak in western markets in 2017 as they enter new phase of maturity. Available on http://tiny.cc/ih0y6x.

Chakrabarti, S., Cox, E., Frank, E., Gting, R. H., Han, J., Jiang, X., Kamber, S. S., Nadeau, T. P., Neapolitan, R. E., Pyle, D., Refaat, M., Schneider, M., Teorey, T. J., and Witten, I. H. (2008). *Data Mining: Know It All*. Morgan Kaufmann Publishers Inc.

Chen, M., Zhang, Y., Li, Y., Mao, S., and Leung, V. C. M. (2015). Emc: Emotion-aware mobile cloud computing in 5g. *IEEE Network*, **29**(2), 32–38.

Chen, S., Wang, Y., and Pedram, M. (2014). Optimal offloading control for a mobile device based on a realistic battery model and semi-markov decision process. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, pages 369–375, Piscataway, NJ, USA. IEEE Press.

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., and Patti, A. (2011a). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA. ACM.

Chun, S., Maniatis, P., Naik, M., and Patti, A. (2011b). Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of the Sixth Conf. on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA. ACM.

Costa, I., Araujo, J., Dantas, J., Campos, E., Silva, F. A., and Maciel, P. (2015). Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Quality and Reliability Engineering International*, pages n/a–n/a.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th Int. Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA. ACM.

da Silva, V. C. O., Oliveira, D. M., de Araujo, J. C. T., and Maciel, P. R. M. (2014). Energy consumption in mobile devices considering communication protocols. *Advances in Information Sciences and Service Sciences*, **6**(5), 1.

Desrochers, A., Al-Jaar, R., and Society, I. C. S. (1995). *Applications of petri nets in manufacturing systems: modeling, control, and performance analysis*. IEEE Press.

Dey, S., Liu, Y., Wang, S., and Lu, Y. (2013). Addressing response time of cloud-based mobile applications. In *Proc. of the First Int. Workshop on Mobile Cloud Computing & Networking*, MobileCloud, pages 3–10, New York, USA. ACM.

Efron, B. and Tibshirani, R. (1993). *An Introduction to the Bootstrap*. Chapman and Hall.

Eom, H., St Juste, P., Figueiredo, R., Tickoo, O., Illikkal, R., and Iyer, R. (2012). Snarf: a social networking-inspired accelerator remoting framework. In *Proc. of the first edition of the MCC workshop on Mobile cloud computing*, pages 29–34. ACM.

Flores, H. and Srirama, S. (2013). Mobile code offloading: Should it be a local decision or global inference? In *Proceeding of the 11th Annual Int. Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 539–540, New York, NY, USA. ACM.

Gabner, R., Schwefel, H.-P., Hummel, K., and Haring, G. (2011). Optimal model-based policies for component migration of mobile cloud services. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 195–202.

German, R. (2000). *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA.

German, R., Kelling, C., Zimmermann, A., and Hommel, G. (1995). Timenet: a toolkit for evaluating non-markovian stochastic petri nets. *Performance Evaluation*, **24**(1), 69–87.

Gomes, C. N. (2012). *Estudo do Paradigma Computação em Nuvem*. Ph.D. thesis, INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA.

González-Rodríguez, G., Colubi, A., and Gil, M. A. (2012). Fuzzy data treated as functional data: A one-way anova test approach. *Comput. Stat. Data Anal.*, **56**(4), 943–955.

Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 93–106, Berkeley, CA, USA. USENIX Association.

Guimarães, A. P., Maciel, P. R., and Matias Jr, R. (2013). An analytical modeling framework to evaluate converged networks through business-oriented metrics. *Reliability Engineering & System Safety*.

Haverkort, B. R. (2002). *Lectures on formal methods and performance analysis*, chapter Markovian models for performance and dependability evaluation, pages 38–83. Springer-Verlag New York, Inc., New York, NY, USA.

Herzog, U. (2001). *Formal Methods for Performance Evaluation*, pages 1–37. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hong, J. I. and Landay, J. A. (2001). An infrastructure approach to context-aware computing. *Human-Computer Interaction*, **16**(2), 287–303.

Huang, D., Zhang, X., Kang, M., and Luo, J. (2010). Mobicloud: Building secure cloud framework for mobile computing and communication. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 27–34.

Huerta-Canepa, G. and Lee, D. (2010). A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 6:1–6:5, New York, NY, USA. ACM.

Jain, A., Klare, B., and Park, U. (2011). Face recognition: Some challenges in forensics. In *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE Int. Conference on*, pages 726–733.

Jain, R. (2008). *The Art Of Computer Systems Performance Analysis: Techniques for Experimental Measurement, Simulation and Modeling*. Wiley India Pvt. Ltd.

JavaCV (2015). Javacv. https://github.com/bytedeco/javacv. Accessed: 2015-07-28.

Junior, M. N. O., Neto, S., Maciel, P., Lima, R., Ribeiro, A., Barreto, R., Tavares, E., and Braga, F. (2006). *Analyzing Software Performance and Energy Consumption of Embedded Systems by Probabilistic Modeling: An Approach Based on Coloured Petri Nets*, pages 261–281. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. (2012). Cuckoo: A computation offloading framework for smartphones. In M. Gris and G. Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–79. Springer Berlin Heidelberg.

Khan, A., Othman, M., Madani, S., and Khan, S. (2014). A survey of mobile cloud computing application models. *Communications Surveys Tutorials, IEEE*, **16**(1), 393–413.

Khan, A. N., Mat Kiah, M. L., Madani, S. a., Khan, A. U. R., and Ali, M. (2013). Enhanced dynamic credential generation scheme for protection of user identity in mobile-cloud computing. *The Journal of Supercomputing*, **66**(3), 1687..1706.

Khan, u. R. A., Othman, M., Khan, A. N., Abid, S. A., and Madani, S. A. (2015). Mobibyte: An application development model for mobile cloud computing. *Journal of Grid Computing*, **13**(4), 605–628.

Kleinrock, L. (1975). *Queueing Systems*, volume 1. Wiley, New York.

Kocjan, P. and Saeed, K. (2012). Face recognition in unconstrained environment. In K. Saeed and T. Nagashima, editors, *Biometrics and Kansei Engineering*, pages 21–42. Springer New York.

Kosta, S. and Aucinas, e. a. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proc. IEEE*, pages 945–953.

Kovachev, D., Cao, Y., and Klamma, R. (2011). Mobile cloud computing: A comparison of application models. *CoRR*, **abs/1107.4940**.

Kristensen, M. (2010). Scavenger: Transparent development of efficient cyber foraging applications. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 217–226.

Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. (2013). A survey of computation offl.g for mobile systems. *Mob. Netw. Appl.*, **18**(1), 129..140.

Kuo, W. and Zuo, M. J. (2003). *Optimal reliability modeling: principles and applications*. John Wiley & Sons.

Li, D., Hao, S., Halfond, W. G. J., and Govindan, R. (2013). Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 78–89, New York, NY, USA. ACM.

Lin, X., Wang, Y., Xie, Q., and Pedram, M. (2015). Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Transactions on Services Computing*, **8**(2), 175–186.

Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., and Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, **48**, 99 – 117.

Liu, Q., Jian, X., Hu, J., Zhao, H., and Zhang, S. (2009). An optimized solution for mobile environment using mobile cloud computing. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, pages 1–5.

Luo, R. and Liu, H.-H. (2010). Design and implementation of efficient hardware solution based sub-window architecture of haar classifiers for real-time detection of face biometrics. In *Mechatronics and Automation (ICMA), 2010 Int. Conference on*, pages 1563–1568.

Ma, X., Huang, P., Jin, X., Wang, P., Park, S., Shen, D., Zhou, Y., Saul, L. K., and Voelker, G. M. (2013). edoctor: Automatically diagnosing abnormal battery drain issues on smartphones. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 57–70, Berkeley, CA, USA. USENIX Association.

Maciel, Paulo R. M.; Trivedi, K. S. M. R. J. and Kim, D. S. (2011). Performance and dependability in service computing: Concepts, techniques and research directions. **1**(1), 53–97.

Maciel, P., Trivedi, K. S., Matias, R., and Kim, D. S. (2011). *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, chapter Dependability Modeling. Premier Reference Source. Igi Global.

Malhotra, M. and Reibman, A. (1993). Selecting and implementing phase approximations for semi-markov models. *Stochastic Models*, **9**(4), 473–506.

Marsan, M. A. (1990). Advances in petri nets 1989. chapter Stochastic Petri Nets: An Elementary Introduction, pages 1–29. Springer-Verlag New York, Inc., New York, NY, USA.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. (1994). *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

Matos, R., Araujo, J., Oliveira, D., Maciel, P., and Trivedi, K. (2015). Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simulation Modelling Practice and Theory*, **50**, 151–164.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing.

Mendonca, J., Lima, R., Andrade, E., and Callou, G. (2015). Assessing performance and energy consumption in mobile applications. In *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, pages 74–79.

Miller, M. (2008). *Cloud computing: Web-based applications that change the way you work and collaborate online*. Que publishing.

Molloy, M. K. (1982). Performance analysis using stochastic petri nets. *IEEE Trans. Comput.*, **31**, 913–917.

Montgomery, D. C. and Montgomery, D. C. (1984). *Design and analysis of experiments*, volume 7. Wiley New York.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, **77**(4), 541–580.

N. Nikolaidis, P. N. (2002). *Instruction-level Power Measurement Methodology*. Electronics Lab, Physics Dept, Aristotle University of Thessaloniki Greece.

Nagle, J. B. (1988). Innovations in internetworking. chapter On Packet Switches with Infinite Storage, pages 136–139. Artech House, Inc., Norwood, MA, USA.

Nelson, R. (2013). *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer Science & Business Media.

Nimmagadda, Y., Kumar, K., Lu, Y.-H., and Lee, C. (2010). Real-time moving object recognition and tracking using computation offloading. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2449–2455.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In *CCGRID '09*, pages 124–131.

Oliveira, D., Araujo, J., Matos, R., and Maciel, P. (2013). Availability and energy consumption analysis of mobile cloud environments. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4086–4091. IEEE.

Olteanu, A.-C. and Tapus, N. (2013). Offloading for mobile devices: A survey.

OpenCV (2015). Opencv. http://opencv.org/. Accessed: 2015-07-28.

Ou, S., Yang, K., Liotta, A., and Hu, L. (2007). Performance analysis of offloading systems in mobile wireless environments. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1821–1826. IEEE.

Pandey, S. and Nepal, S. (2012). Modeling availability in clouds for mobile computing. In *2012 IEEE First International Conference on Mobile Services*, pages 80–87.

Park, J., Yu, H., Chung, K., and Lee, E. (2011). Markov chain based monitoring service for fault tolerance in mobile cloud computing. In *Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, WAINA '11, pages 520–525, Washington, DC, USA. IEEE Computer Society.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proc. of the 12th Conf. on Evaluation and Assessment in Software Engineering*, EASE'08, page 68..77, Swinton, UK, UK. British Computer Society.

Pitkänen, M., Kärkkäinen, T., Ott, J., Conti, M., Passarella, A., Giordano, S., Puccinelli, D., Legendre, F., Trifunovic, S., Hummel, K. A., May, M., Hegde, N., and Spyropoulos, T. (2012). SCAMPI: Service platform for social aware mobile and pervasive computing. In *MCC 2012, ACM Mobile Cloud Computing Workshop, collocated with ACM Sigcomm, August 17, 2012, Helsinki, Finland / Also published in SIGCOMM Computer Communication Review , Volume 42 Issue 4, September 2012*, Helsinki, FINLAND.

PowerTutor (2014). A power monitor for android-based mobile platforms. Available on http://ziyang.eecs.umich.edu/projects/powertutor/.

Pungila, C. and Negru, V. (2012). A highly-efficient memory-compression approach for gpu-accelerated virus signature matching. In *Information Security*, Lecture Notes in Computer Science, pages 354–369.

Rahimi, M. R., Venkatasubramanian, N., Mehrotra, S., and Vasilakos, A. V. (2012). Mapcloud: Mobile applications on an elastic and scalable 2-tier cloud architecture. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 83–90, Washington, DC, USA. IEEE Computer Society.

Reduction, C. (2015). Colour reduction. http://tinyurl.com/pwq8j44. Accessed: 2015-07-28.

Saranya, S. M. and Vijayalakshmi, M. (2011). Interactive mobile live video learning system in cloud environment. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 673–677. IEEE.

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

Se, S., Barfoot, T., and Jasiobedzki, P. (2005). Visual motion estimation and terrain modeling for planetary rovers. In *Proceedings of the International Symposium on Artificial Intelligence for Robotics and Automation in Space*.

Shi, S., Hsu, C.-H., Nahrstedt, K., and Campbell, R. (2011). Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 103–112, New York, NY, USA. ACM.

Shih, C.-S., Chen, Y.-K., Chen, J., and Chang, N. (2013). Virtual cloud core: Opencl workload sharing framework for connected devices. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th Int. Symposium on*, pages 486–493.

Silva, B., Callou, G., Tavares, E., Maciel, P., Figueiredo, J., Sousa, E., Araujo, C., Magnani, F., and Neves, F. (2013). Astro: An integrated environment for dependability and sustainability evaluation. *Sustainable Computing: Informatics and Systems*, **3**(1), 1 – 17.

Silva, B., Tavares, E., Maciel, P., Nogueira, B., Oliveira, J., Damaso, A., and Rosa, N. (2014a). Amalghma -an environment for measuring execution time and energy consumption in embedded systems. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3364–3369.

Silva, B., Tavares, E., Maciel, P., Nogueira, B., Oliveira, J., Damaso, A., and Rosa, N. (2014b). Amalghma -an environment for measuring execution time and energy consumption in embedded systems. In *SMC Conference*, pages 3364–3369.

Silva, F. A., Maciel, P., Quesado, E., Germano Zaicaner, M. D., and Silva, B. (2015a). Benchmark applications used in mobile cloud computing: A systematic mapping study. *The Twentieth IEEE Symposium on Computers and Communications (ISCC)*.

Silva, F. A., Rodrigues, M., Maciel, P., Kosta, S., and Mei, A. (2015b). Planning mobile cloud infrastructures using stochastic petri nets and graphic processing units. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 471–474.

Silva, F. A., Maciel, P., Alves, G., and Matos, R. (2015c). A scheduler for mobile cloud based on weighted metrics and dynamic context evaluation. In *Applied Computing (SAC 2015), Proc. of The 30th ACM/SIGAPP Symposium On*.

Silva, F. A., Maciel, P., and Matos, R. (2015d). Smartrank: A smart scheduling tool for mobile cloud computing. *J. Supercomput.*, **71**(8), 2985–3008.

Silva, F. A., Zaicaner, G., Quesado, E., Dornelas, M., Silva, B., and Maciel, P. (2016a). Benchmark applications used in mobile cloud computing research: a systematic mapping study. *The Journal of Supercomputing*, **72**(4), 1431–1452.

Silva, F. A., Maciel, P., Santana, E., Matos, R., and Dantas, J. (2016b). Mobile cloud face recognition based on smart cloud ranking. *Computing*, pages 1–25.

Silvaa, B., Maciela, P. R. M., Zimmermannb, A., and Brilhantea, J. (2014). Survivability evaluation of disaster tolerant cloud computing systems.

Soyata, T., Muraleedharan, R., Funai, C., Kwon, M., and Heinzelman, W. (2012). Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000059–000066.

Stewart, W. J. (1994). *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press.

Subramanian, K. (2011). Hybrid clouds. *Access from http://emea. trendmicro. com/imperia/md/content/uk/cloud-security/wp01_hybridcloud-krish_110624us. pdf*.

Tavares, E., Maciel, P., Silva, B., Oliveira, M., and Rodrigues, R. (2007). Modelling and scheduling hard real-time biomedical systems with timing and energy constraints. *Electronics Letters*, **43**(19), 1015–1017.

Tavares, E., Maciel, P., Dallegrave, P., Silva, B., Falcão, T., Nogueira, B., Callou, G., and Cunha, P. (2010). Model-driven software synthesis for hard real-time applications with energy constraints. *Des. Autom. Embedded Syst.*, **14**(4), 327–366.

Terry, D. (2011). Acm tech pack on cloud computing. *ACM Tech Pack Committee on Cloud Computing*.

Triola., M. (2004). *Elementary Statistics*. Addison Wesley, 9 edition.

Trivedi, K. S. (2001). *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley and Sons, New York.

Trivedi, K. S. (2002). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons Ltd., Chichester, UK, 2nd edition edition.

Turk, M. and Pentland, A. (1991). Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition Proc. CVPR, IEEE Computer Society Conf. on*, pages 586–591.

Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *J. Comput. Vision*, **57**(2), 137–154.

WattsUp (2016). Watts up: Monitor real-time electricity usage. Available on http://www.inds.co.uk/product/watts-up-pro/.

Xing, T., Liang, H., Huang, D., and Cai, L. (2012). Geographic-based service request scheduling model for mobile cloud computing. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th Int. Conference on*, pages 1446–1453.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, **1**(1), 7–18.