Novel Application Models and Efficient Algorithms for Offloading to Clouds

by

José Andrés González Barrameda

Thesis submitted in partial fulfillment of the requirements for the
Doctorate in Philosophy degree in Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The application offloading problem for Mobile Cloud Computing aims at improving the mobile user experience by leveraging the resources of the cloud. The execution of the mobile application is offloaded to the cloud, saving energy at the mobile device or speeding up the execution of the application. We improve the accuracy and performance of application offloading solutions in three main directions. First, we propose a novel fine-grained application model that supports complex module dependencies such as sequential, conditional and parallel module executions. The model also allows for multiple offloading decisions that are tailored towards the current application, network, or user contexts. As a result, the model is more precise in capturing the structure of the application and supports more complex offloading solutions. Second, we propose three cost models, namely, average-based, statistics-based and interval-based cost models, defined for the proposed application model. The average-based approach models each module cost by the expected cost value, and the expected cost of the entire application is estimated considering each of the three module dependencies. The novel statistics-based cost model employs Cumulative Distribution Function (CDFs) to represent the costs of the modules and of the mobile application, which is estimated considering the cost and dependencies of the modules. This cost model opens the doors for new statistics-based optimization functions and constraints whereas the state of the art only support optimizations based on the average running cost of the application. Furthermore, this cost model can be used to perform statistical analysis of the performance of the application in different scenarios such as varying network data rates. The last cost model, the interval-based, represents the module costs via intervals in order to addresses the cost uncertainty while having lower requirements and computational complexity than the statistics-based model. The cost of the application is estimated as an expected maximum cost via a linear optimization function. Finally, we present offloading decision algorithms for each cost model. For the average-based model, we present a fast optimal dynamic programming algorithm. For the statistics-based model, we present another fast optimal dynamic programming algorithm for the scenario where the optimization function meets specific properties. Finally, for the interval-based cost model, we present a robust formulation that solves a linear number of linear optimization problems. Our evaluations verify the accuracy of the models and show higher cost savings for our solutions when compared to the state of the art.

Acknowledgements

I am deeply grateful to my supervisor Dr. Nancy Samaan for her excellence, guidance and support provided throughout all these years, both on the technical and personal levels. With her constant source of knowledge, challenges and constructive feedback, Professor Samaan always created a great balance of motivation for excellence and research freedom.

I frequently reached a special source of knowledge, mainly for the statistical aspects of this work: my brother Hector, my sister Yohanna, and my friend Chiro, while my father provided me with valuable insights on how to conduct research; thank you. I also thank Danial for his friendship during these years as PhD students.

Special people inspired me every day with their dedication under difficult circumstances: my mother raising the family as a single mother throughout the 1990s in Cuba, while still being an outstanding mathematician, my close friend Luis Enrique with his PhD and two MSc in La Habana, and my father and his PhD in the harsh winters of Moscow.

Eternal thanks to my mother and my entire family for being the ideal environment to pursue a goal that requires so much stability and peace for years, to all my nephews for their immense love, and to Marion for her support, specially during the last weeks of writing. I thank my close friends, my brothers, for their constant encouragement: Alain, Arturo, Alexander and Rene.

Last but not least, I thank Professor Tony Mesa for his legacy as a founder of the Computer Science program at Universidad de La Habana.

To my Mother, Father and Nephews

Table of Contents

List of Figures

List of Tables

Glossary

Application model Abstraction to model the mobile application.

CGM The Call Graph Model (CGM) is an application model where the application is represented by a directed acyclic graph whose nodes correspond to the modules of the application and the directed links represent the possible calls between the modules. xiii, 4, 21–23, 29, 43, 46, 50, 54, 119

EDT Execution Dependency Tree (EDT) is an application model where the application is represented by a tree graph where the nodes correspond to the modules of the application and the dependencies between them. xiii, xiv, 5–8, 44, 52, 54, 55, 57, 58, 60, 67, 73–80, 82, 83, 92, 94, 95, 101, 106, 109, 116–118, 120, 121

SGM The Star Graph Model (SGM) is an application model where the application is modelled as a set of independent modules. 20–23, 29, 31, 32, 38, 39, 46, 50

TCM The Thin Client Model (TCM) is an application model where the application is modelled as a single module. When used in the application offloading problem, either the entire application runs on the mobile or is offloaded to the cloud. 19, 23

WGM The Workflow Graph Model (WGM) is an data-centric application model where the application is modelled by a directed workflow graph whose nodes correspond to the steps or processes that are applied to the data, and a link from one node to another represent consecutive steps when processing the data. 22, 23

C-RAN Cloud Radio Access Networks. 17, 40

Cloudlet Architecture for Mobile Cloud Computing (MCC) where the mobile device accesses a server that is deployed nearby, possibly on the same LAN. The local server leverages additional resources provided by a remote cloud. viii, 2, 10–12, 17, 37, 50

Cost model Abstraction to model and estimate the costs the application and its modules. 3–7

Average-based The costs of the modules and of the application are modelled by a single value that represents the expected cost. ii, 4, 6–8, 52, 57, 118–121

Interval-based The costs of the modules are modelled by intervals, the cost of the application is estimated by a single value that represents, for example, the maximum cost. ii, xi, 7, 8, 92, 101, 118–120

Statistics-based The costs of the modules and of the application are modelled by CDFs. ii, ix, 6–8, 65, 66, 79, 80, 84, 118–120

Dependency Relationship between the execution of a set of modules. xii, 18, 23

Parallel All the modules in the set run in parallel. It is a type of node in the Execution Dependency Tree (EDT) model. 18, 23, 53, 118

Probabilistic Only one module in the set runs, according to a probability function. It is a type of node in the EDT model. 18, 53, 106, 118

Series All the modules in the set run in sequence. It is a type of node in the EDT model. xiv, 18, 23, 53, 118

Execution path Sequence of modules calls leading to the call of a module. It corresponds to the state of the call stack of the application execution at the time of the call of the module. 52

FD Application that performs face detection and whose EDT is shown in Figure 3.3.

fd0 Module in the FD application that performs the face-detection operation when called from the main module. ix, 60–62, 82, 88–90, 101, 103, 104, 106, 109

fd1 Module in the FD application that performs the face-detection operation when called from the th module. xiii, 60–62, 64, 83, 101, 103, 104

main Module in the FD application that runs when the application is started. xiii, 60, 103, 104

rz Module in the FD application that is used to resize an image. xiii, 60, 83, 101, 103, 104

th Module in the FD application that performs the face-detection operation by first calling module rz to resize the image and then calling node fd1 to performing the face-detection operation. xiii, 60, 83, 101, 103, 104

LP Linear Programming (LP) refers to optimization problems where the objective function and the constraints are expressed as linear combinations of the variables. 4, 8, 29, 30, 46, 47, 50, 114, 116

BLP Binary Linear Programming (BLP), or 0–1 Integer Linear Programming, is a linear programming optimization problem where the variables are restricted to be binary. xiii, 44, 48, 98, 119

ILP Integer Linear Programming (ILP) is a linear programming optimization problem where all of the variables are restricted to be integers. 43, 47

MAUI Solution for the application offloading problem proposed in [2]. The application is modeled by a Call Graph Model (CGM) and the problem is formulated as a Binary Linear Programming (BLP) problem. 4, 17, 24, 32, 36, 41, 44

MEC Mobile Edge Computing (MEC) is an architecture for MCC where the mobile device leverages resources from a cloud that is located at the cellular infrastructure providing low latency and ubiquitous access to cloud resources. viii, xiv, 12, 13

Mobile Cloud Architecture for MCC where the mobile devices act both as clients and servers, and are connected via P2P protocols and without a dedicated infrastructure. This architecture is also known as Cirrus Clouds [3], P2P Cloud, and Mobile Grid [4]. viii, 2, 10, 11, 15–17, 29, 30, 32, 33, 36, 41, 50

Nested execution Refers to the execution of a module that invokes or triggers the execution of yet another module. 18, 23

Remote Cloud Architecture for MCC where the mobile device leverages resources from a cloud that is located in different network than the mobile device and that is geographically distant. viii, 2, 10, 49

Robust optimization Robust optimization refers to optimization problems where the input parameters are uncertain and take values from uncertainty sets. Typically, a solution is feasible if it meets the constraints for any realization of the parameters. Similarly, a solution is optimal if it is feasible and it provides the minimum guaranteed value for all the realizations of the parameters within the uncertainty sets. 7, 8, 45, 47–49, 51, 96, 97, 116, 119

Single-Series Refers to applications that can be represented by an EDT with exactly one series dependency and no other dependency. x, 101, 104, 105, 109–114

Chapter 1

Introduction

Mobile Cloud Computing has attracted great research interest due to the astonishing increase of the number of mobile users in recent years. In order to address the growing expectations of mobile users, the mobile applications leverage cloud resources using increasingly efficient network communication protocols. In this chapter, we present an overview of Mobile Cloud Computing and the application offloading problem, followed by a discussion of the motivation, main objectives and contributions of this work.

1.1 Mobile Cloud Computing and Application Offloading

The global net number of new mobile devices with advanced computing and multimedia capabilities, and with a minimum of 3G connectivity, was 497 and 563 millions in 2014 and 2015, respectively [5, 6]. This trend is expected to continue reaching 11.5 billion of mobile devices by 2019 [5, 6]. On the one hand, these devices have limited resources such as battery capacity, CPU power and memory. This limitation is even more apparent when compared to the amount of resources available as part of Cloud Computing (CC). On the

other hand, the latest developments in network communication standards such as LTE, have resulted in an increase of 20 percent of the speed of the cellular connection worldwide in 2015 [6], and thus have brought the mobile device closer to cloud resources. These facts make Mobile Cloud Computing both a great demand and a reality nowadays.

Mobile Cloud Computing aims at improving the experience of the mobile user by leveraging resources from the cloud. Using MCC, the user perceives more responsive mobile applications, with lower energy footprint on the mobile device. These applications are also able to access and process large data sets that are stored far from the mobile device, and to cooperate with other mobile users to complete complex tasks.

Mobile Cloud Computing is envisioned in three main architectures. In *Remote Cloud*, the mobile device accesses a cloud server that is hosted in a remote location, using WLAN or cellular network interfaces. One or more cloud servers are chosen from large server farms based on the location of the mobile device. Example of realizations of this architecture are Google Drive [7] and Amazon EC2 [8], which have data and processing centers with thousands of servers in several locations around the globe. Alternatively, a design called *Cloudlet* [9] proposes to deploy servers near mobile users, such as at a coffee shop. The proximity reduces the communication cost between the mobile device and the local server, while the local server can leverage resources from the remote cloud if needed. Finally, a group of mobile devices may choose to cooperate and form a *Mobile Cloud*. The mobile devices form a peer-to-peer network and behave both as clients and servers, providing resources to other mobile devices.

There are multiple challenges concerning the development of the mobile cloud computing technology. The implementation and adoption of the technology by users must consider risks such as security, performance unpredictability, the use of proprietary software by the cloud provider and the possibility of the cloud provider going out of business. Mobility management is another concern, specially for Mobile Clouds, as the mobility of the user may result in changes to the topology of the network. Hence, new algorithms and protocols must adapt to these changes. These protocols must also consider the communication technology, (e.g., WLAN and Bluetooth), which can affect the performance of the mobile applications. For instance, WLAN usually offers higher speed and energy efficiency compared to cellular networks but has a smaller area coverage. There are additional challenges related to the application offloading problem.

The application offloading problem consists in running parts of or the entire mobile application in surrogate servers in the cloud in order to minimize the total cost of running the application. The decision must evaluate the tradeoff between the resources saved on

the device by performing computations on the server, and the resources used when the mobile device and the server communicate. This problem poses unique challenges such as choosing an execution offloading model, modeling of the mobile application, estimating the application running cost for a given offloading decision, and developing an algorithm to find the offloading decision that minimizes the cost of running the application.

The offloading operation is usually performed via client-server protocols or via VM-migration. In the client-server approach, the modules of the application are offloaded to the server via well established protocols such as RPC. The input parameters and global variables of the application execution are sent to the server where the module is executed. The result of the computation is sent back to the mobile device. This method is fine-grained but requires modifying the application. VM-migration offloads the entire state of application to a VM that emulates the same environment of the mobile device and that is running on the server. The application is loaded in the VM where its execution resumes, perceiving a system with augmented resources. This method requires none or almost no changes to the application by the developer. However, it results in higher delays due to the time consumed in capturing and transferring the image of the mobile from the mobile device to the server and back.

A key aspect in application offloading to efficiently estimate the execution costs and make the offloading decision is to accurately model the application. Conceptually, the application is divided into modules that can be offloaded or not according to their associated execution costs at both the clouds and the device.

The application is often represented by a directed graph where each node is a module of the application and a link represents a call between two nodes [2, 10, 11]. Figure 1.1 depicts an example of this model, referred to as Call Graph Model. Another approach is to consider that the modules are independent, in other words, there are no calls from one module to another. This model is used often for its simplicity, and is appropriate for simple applications and service-oriented architectures, but it is not accurate for general applications because it does not captures the dependency relation between modules. A third type of model uses workflow graphs to represent the application; it focuses on the data and the transformations that it is subject to. The application models constitutes the basis for the second step in application offloading, which is the construction of an efficient cost model.

The cost model is used to estimate the cost of running the application, evaluating the tradeoff of resources saved and consumed during the offloading operations. The costs of executing each module on the mobile device and on the cloud, as well as the communica-

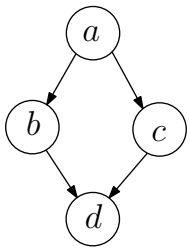


Figure 1.1: A Call Graph Model of an application with four modules, a, b, c and d. Node a calls nodes b and c, and they both call d.

tion cost of offloading the module, are represented by values that represent energy, time, monetary cost, etc. The expected running cost of the entire application is defined as a function of these costs and it is used as the objective function of the offloading optimization problem. In an alternative cost model, the modules represent methods in object oriented programming, and the developer specifies the expected cost of the method as a function of the input parameters of the method [12].

The goal of the offloading decision algorithm is to find the optimal execution location for each module of the application such that the cost of executing the application is minimized. Some approaches use Linear Programming (LP) formulations while other develop heuristic algorithms. Some find the offloading decision of all the modules at once, or implement scheduling algorithms that decide to offload or not to offload dynamically when the module must be run. While the application offloading problem is proven to be NP-hard when using Call Graph Model (CGM) [13], instances of the problem are not necessarily large and exhaustive algorithms could be appropriate in some scenarios.

1.2 Motivation

We consider the CGM and the average-based cost model as the most advanced models used so far for application offloading. These models, however, have three main limitations. First, as the execution cost of a module is estimated as an average cost, optimality and feasibility of offloading solutions can only be defined in terms of average costs. Yet, for a real time application for example, we may want to define the feasibility in terms of bounded maximum application cost, instead of bounded average cost. For example, MAUI [2] proposes an offloading method that aims at finding a solution that results in minimum energy cost while the time cost of the application is bounded. However, as MAUI uses an average-based cost model, the application running time can be above said bound in some executions of the application, and thus not feasible. The same can be said regarding

optimality, i.e., while the average energy consumption is minimized, some executions of the application may still consume an amount of energy well above the average.

Second, the application model represents each module by exactly one node in the graph. This typically limits the cost model to only one cost estimate per module, and the offloading decision to one decision per module. However, the cost of a module may exhibit different costs in different executions, and so it may be better to offload some executions of the same module to the cloud while running other executions on the mobile device.

Finally, the application and cost models do not take into account the different programming instructions, such as conditionals or parallel executions. Yet, these instructions determine the execution cost of the application. For example, the total cost of the application that executes two modules is different if both modules are run in parallel or sequentially.

1.3 Thesis Statement

The main goal of this thesis is to present and promote novel accurate and efficient algorithms for application offloading for MCC. This goal is achieved by creating novel application and cost models that accurately capture the structure and cost of the mobile application. Based on these models, we present novel formulations of the application offloading problem and propose new offloading decision algorithms.

1.4 Contributions

This thesis makes the following contributions.

1.4.1 Fine Grained Application Model

We present a novel fine grained application model called Execution Dependency Tree (EDT) that uses a tree graph to model the application. This model has two main advantages over the models of the state of the art. First, each module of the application is represented by multiple nodes in the tree according the different execution paths leading to the execution of the module. Second, the model captures sequential, conditional, and parallel module executions via *series*, *probabilistic* and *parallel* dependency nodes, respectively.

As a result, the proposed model supports fine grained cost models and offloading decisions. This application model has been published in [14].

1.4.2 Cost Models

We have developed three cost models that are based on the proposed application model.

Average-based Cost Model for the Novel Application Model

We propose a cost model that uses the average of each cost for each module. The cost model permits to estimate the average execution cost of running the entire application. While using the average cost of the modules is not new, the novelty of the proposed model is to estimate the expected application cost according to the three module dependencies supported by the EDT application model. This results in a simple and effective cost model for quickly finding average cost estimates and offloading decisions for the mobile application. This cost model has been published together with the application model in [14].

Statistical Cost Model

We improve existing average-based cost models by proposing a statistics-based cost model where the execution cost of each module is represented by a random variable with a CDF. The running cost of the entire application is estimated statistically via a CDF that is computed from the costs of the modules and the dependencies described by the application model. The model has two main benefits. First, it is the first, to our knowledge, that supports statistical optimization functions and constraints other than just the average application cost. Second, the cost model is the first that can be used to perform statistical analysis of the application execution costs for different scenarios, such as network speeds or the CPU speed of the mobile devices. These results have been published in [15].

Interval-based Cost Model

We present an interval-based cost model where each cost is modeled by an interval. Typically, the lower and upper bounds for each module cost will represent the mean and the maximum possible values of said cost, respectively. The cost of the entire application is estimated as single value that represents the worst case cost of the application. This

cost model is not limited to this application cost estimate as we will see when we present the problem formulation using this model. This approach addresses scenarios where the offloading decision must provide some guarantees to uncertainty on the application cost but where the statistics-based model cannot be used. Cases where the statistics-based approach cannot be used include when there is not enough data to accurately estimate the CDFs of the costs, or when a more tractable cost model is needed. The main benefits of the model can be summarized as (1) a tractable approach to (2) address uncertainty in the application costs, (3) providing guarantees on the application costs. This approach have been submitted for publication in [16].

1.4.3 Efficient Offloading Decision Algorithms

We develop efficient optimal decision algorithms based on the EDT model for the three proposed cost models.

Algorithm using the Average-based Cost Model

We develop an algorithm that minimizes the average application cost using the average-based cost model. The algorithm is efficient and optimal and follows a dynamic programming approach. This algorithm has been published in [14].

Algorithm using the Statistical Cost Model

For the statistics-based cost model, we develop a fast optimal algorithm based on dynamic programming for the particular case where the statistical measurement function meets certain properties. The main benefits of the algorithm is being efficient and optimal for that particular scenario. We also discuss the general problem of minimizing any statistical measurement function of the application's cost, such as the 95 percentile. The algorithms for the statistics-based cost model has been published in [15].

Algorithm using the Interval-based Cost Model

For the interval-based cost model, we propose a robust optimization formulation of the application offloading problem that aims at minimizing the application cost when a given number of costs may take the worst case in its interval. In other words, the solution guarantees a minimum application cost when a given number of parameters are uncertain.

The robust problem is solved solving a polynomial number of LP problems. This method is flexible in that it supports optimization problems that range from minimization of the average application cost to minimization of the worst case application cost. The algorithm for the robust formulation using the interval-based cost model has been submitted in [16].

1.4.4 Publications

These contributions have been published or submitted for publication as follows.

- The novel EDT Application Model and Average-based Cost Model have been published in [14].
- The novel Statistics-based Cost Model together with the novel use cases have been published in [15].
- The robust approach using the interval-based cost model and robust offloading decision algorithm have been submitted to [16].

1.5 Outline

The remainder of the thesis is organized as follows.

- Chapter 2 provides an overview of MCC and discusses the application offloading problem with its most relevant solutions. The chapter also introduces the robust optimization problem.
- Chapter 3 presents the novel EDT application model, the average-based cost model, and an offloading decision algorithm using said cost model.
- Chapter 4 proposes the novel statistics-based cost model, its new use cases and the offloading decision algorithms using this cost model.
- Chapter 5 proposes the novel interval-based cost model, formulates the application offloading problem as a robust optimization problem, and presents an efficient algorithm to solve the problem.
- Chapter 6 presents a summary of the thesis and outlines future research work.

Chapter 2

Background

In this chapter, we provide an introduction to the basic concepts of Mobile Cloud Computing, illustrating its common architectures and challenges. We then focus on the application offloading problem and discuss its main issues, e.g., application and cost modeling problems and offloading decision algorithms. We complete the chapter with an analysis of several existing solutions and algorithms.

2.1 Mobile Cloud Computing

Mobile Cloud Computing provides an efficient solution to the problem of limited resources, (e.g., energy), in mobile devices, aiming at improving the experience of mobile users. Among the benefits provided to the users are improved battery life, faster application executions, and the ability to access and process a large amount of information instantaneously from the mobile device. This is achieved by carefully leveraging the resources provided by the cloud via the device's network interfaces. For example, data storage services such as Dropbox [17] and Google Drive [7] store files in the cloud and, optionally, on the mobile device. The user works on the files regardless of the location of the files and, as a consequence, the user perceives a device with more storage than what the device actually has.

Likewise, assistant services such as Apple's Siri [18] and Google Assistant [19] run in the cloud and are accessed seamlessly from the mobile device.

This technology integrates the mobile device to the cloud and, naturally, it offers several benefits similar to cloud computing. Some of these benefits are virtually infinite resources, no upfront commitment from users, a pay-per-use cost model, improved resource utilization efficiency, and economy of scale [20]. Other features include adaptability, scalability, availability and self-awareness [21]. Designing support for these features, however, depends on the implemented architecture.

2.1.1 Architectures

There are four main architectures considered for Mobile Cloud Computing [22], namely, *Remote Cloud*, *Mobile Cloud*, *Cloudlet* and *Mobile Edge Computing*. In the first scenario, called *Remote Cloud* and depicted in Figure 2.1, a remote cloud provides resources to the mobile device via the device's network interfaces such as the WLAN and the cellular network interfaces. The cloud server is hosted far from the client and is chosen according to the location of the device. Examples are mobile applications using cloud services such as Google's Gmail [23] and Dropbox [17].

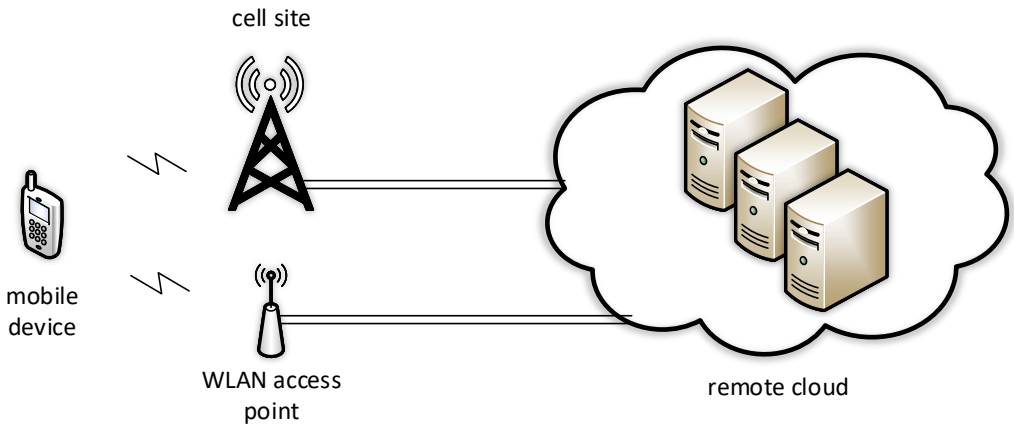


Figure 2.1: Mobile Cloud Computing via a Remote Cloud architecture.

In the second scenario, named *Mobile Cloud* and shown in Figure 2.2, a set of mobile devices cooperate to provide services to other mobile devices. In this scheme, the topology of the cloud may change very often and devices must decide whether to cooperate or not considering that they all have limited resources. The increase in the number of mobile

devices, including laptop, tablet, smartphone and smartwatch, makes it possible to have more powerful and cheaper support for mobile clouds. Huerta et al. [24] show that application offloading in Mobile Cloud is feasible despite initial results that did not achieve an improvement in performance or energy consumption. Other works addressing application offloading for this architecture include [25], [3] and [26]. Litke et al. [4] define the Mobile Cloud as a distributed, high performance heterogeneous infrastructure for computing and data management using mobile devices. The authors identify several challenges such as job scheduling, replication, migration and monitoring, where jobs must be executed by the mobile devices optimizing resource utilization and meeting user constraints. Furthermore, noting that a single user may own multiple mobile devices, Kurdi et al. [27] introduce *personal mobile grids* as the mobile cloud that can be formed by a single user's personal mobile devices. Other names used to refer to this architecture include Mobile Grid [4], P2P clouds and Cirrus Clouds [3].

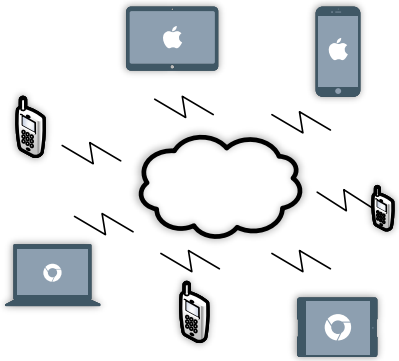


Figure 2.2: Mobile Cloud Computing via a Mobile Cloud architecture.

In the third scenario, referred to as *Cloudlet* and illustrated in Figure 2.3, a local cloud is available to the mobile device via a low-latency, one-hop, high-bandwidth wireless conexion. The approach is proposed by Satyanarayanan et al. in [9] to address the negative effects of latency in offloading solutions and user experience. They argue that latency is likely to worsen as more protocols are added to the communication layers to address issues such as security. Clinch et al. show that the experience of the mobile user is improved by having closer cloud resources instead of offloading to distant clouds [28]. The approach allows mobile users to obtain the benefits of the cloud while avoiding the latency inherent to access the cloud in a remote location. Nevertheless, the local cloud must be connected to a more powerful distant cloud that provides additional resources. Cloudlets are envisioned as clusters of multi-core computers with high Gigabit internal connectivity that are self-

managed in order to require little to no administration. The proposal is characterized as a data center in a box at business premises with decentralized ownership by local business and serving few users [9]. However, due to limited coverage of WLAN networks, this approach cannot guarantee ubiquitous service provision everywhere. Also, the local cloud usually consists of a computing sever or cluster with a small or medium amount of resources, which may not satisfy QoS of a large number of users.

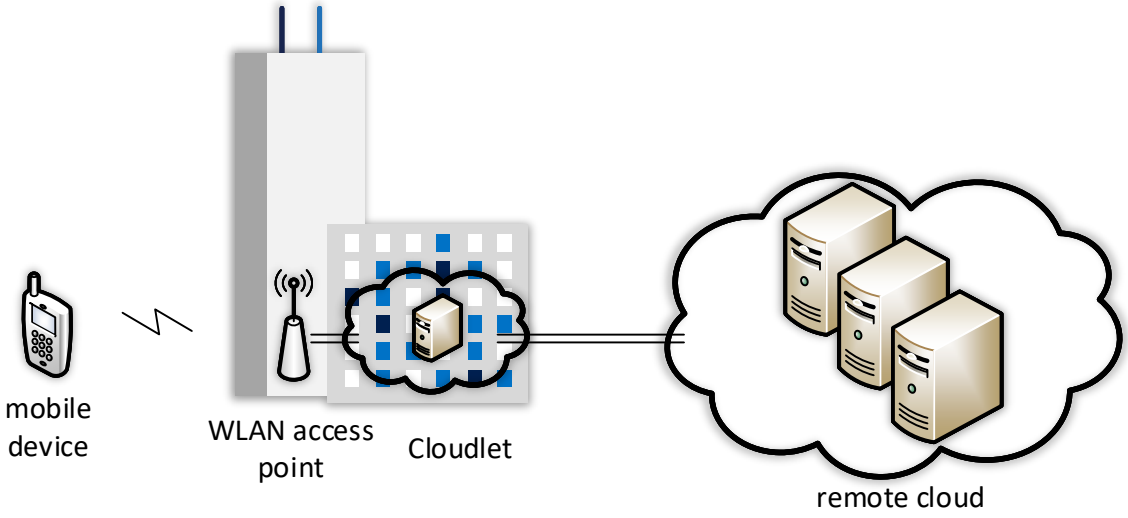


Figure 2.3: Mobile Cloud Computing via a Cloudlet architecture.

The fourth architecture, which has been proposed recently, is Mobile Edge Computing (MEC) [29]. MEC is a mobile cloud computing paradigm that addresses the latency limitation of remote clouds while providing higher range and cloud capacity than Cloudlet-based solutions. As depicted in Figure 2.4, large scale cloud resources are deployed at the edge of pervasive radio access networks, typically by the telecom provider, near the mobile users. These cloud resources may be deployed at multiple locations such as cellular network base stations and multi-Radio Access Technology (RAT) cell aggregation sites. MEC provides not only low latency and high bandwidth, but also real-time access to radio network information that can be leveraged by applications. Some applications that benefit from this architecture include augmented reality and internet of things gateway [29, 30].

2.1.2 Challenges

There are several issues that must be addressed when implementing the mobile cloud computing. In this section, we describe some of these challenges grouped in four categories.

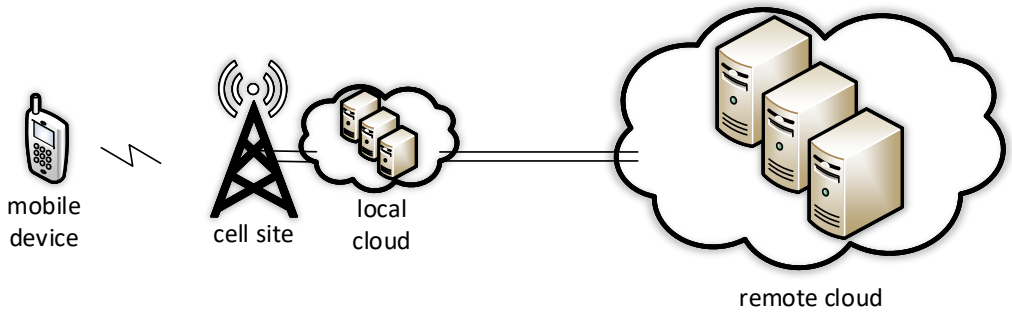


Figure 2.4: Mobile Cloud Computing via a MEC architecture.

Adoption The implementation and adoption of MCC present several issues that are also present in CC. One issue is service availability and business continuity, which refers to the impact on the clients when a cloud provider goes out of business. For instance, a company (the cloud client) offering a service that depends on a cloud provider may be forced to interrupt the service if the provider goes out of business. To overcome this risk, several cloud providers could be used simultaneously. However, cloud providers use proprietary software which makes it difficult to export data between clouds.

Security is another concern because data is not on users premises and therefore data confidentiality and auditability might be difficult to enforce and guarantee. Even more, the cloud data center can be located in a country other than that of the cloud client with different legislation, and the data transfer might cross other countries with yet other privacy laws. Therefore, these countries foreign to the cloud client might access the data based on their own legislation and privacy acts. This has led, for example, to countries prohibiting its government agencies to use cloud providers that store or transfer the data in or through other countries.

Other issues that affect the adoption of cloud computing are data transfer bottlenecks, performance unpredictability, scalable storage, faults in large distributed systems such as Hadoop, and quick scaling [20]. These issues can be handled by a realm of options such as adoption of better technologies, like solid state drives and faster network standards, in order to improve performance unpredictability due to I/O interference, or the creation of a storage system that can scale arbitrary. Further, using virtual machines help in overcoming faults in large-scale distributed systems such as Hadoop [31], and automatic and fast scaling and fine grained cost models save resources, for example Google model of charging per CPU cycle [20].

Mobility Management The protocols for mobile cloud computing must adapt to changes in the topology of the network that may result from user's mobility. One option is for the algorithm to implement a proactive approach where changes are predicted before they occur, trying to avoid possible performance degrading decisions. Another option is to follow a reactive approach where the algorithm adapts and recover from changes only after they occur. It is also possible to have hybrid schemes that try to avoid future-inefficient decisions while also being able to recover from unexpected topology changes.

Proactive location aware algorithms estimate future changes to the topology based on the location and movement of the nodes. They come in two main trends: infrastructure-based and peer-based. On the one hand, infrastructure-based algorithms use GPS, the cellular network and the WLAN to estimate the location of the device. These are precise and provide geographic location, however, they are more demanding in terms of energy and are not always available. On the other hand, peer-based location algorithms use information like signal strength from peer devices, previous peer encounters, accelerometers and compass.

Among reactive algorithms we find *fault tolerant algorithms*. They may rely on fault tolerant platforms for specific tasks, for example relying on Hadoop [31] for file system access, or provide custom recovery mechanisms. Another reactive approach is to support mobility through module and proxy migration. In this scheme, software is composed of modules that are distributed among mobile devices, and of proxies that allow mobile devices to access modules in different servers. According to the location of the mobile devices, their serving proxies and modules are chosen. Then, if the location of the device changes, also their proxies and modules change.

Communication Technology Mobile devices communicate to other devices mainly via three groups of communication standards, namely WLAN, Bluetooth and cellular networks. In terms of performance, using WLAN for MCC provides the best speed and energy gains of all three protocols due to lower RTT and lower energy consumption. Studies show that using WLAN can provide twice the benefits compared to using 3G cellular communication [2]. However, this difference could change as cellular communications are evolving and faster 4G standards are a reality nowadays. Also, WLAN coverage is very small compared to that of cellular networks and is not available in many scenarios. Bluetooth is much slower and has a much lower communication range (about 10 m), but does not require dedicated infrastructure and may be the only solution available in some cases.

End-user challenges From the user's perspective we face two main issues for adoption of MCC under the Mobile Cloud architecture: incentive to communicate, and presentation and usability issues. The reason for user collaboration is to use the resources that are available in other mobile devices, although users may not be willing to sacrifice their resources unless there are incentives for doing so. Incentives can be related to the type of work and how many users need to get the task done. If many users need to achieve a goal, the tasks can be split among them so they contribute with a fraction of the total effort. Another incentive can be to avoid consuming data while roaming. Presentation and usability issues are due to intrinsic characteristics of mobile devices like small screen size, different platforms and input methods.

There are other challenges in mobile cloud computing, in particular concerning the problem of application offloading for mobile cloud computing, which is the main focus of this thesis. We present the problem and discuss its main issues in detail next.

2.2 Application Offloading

Informally, the *Application Offloading* problem refers to that of deciding the location of execution of the mobile application, or parts thereof, such that the execution of the application results in the minimum cost. The cost expresses time, energy, monetary costs or a combination of them, and may vary according to the status of the mobile device's battery or the user preferences. For example, if the energy remaining in the mobile device's battery is low, the goal of the problem could be to extend the battery life by minimizing the energy cost of the application. The location generally refers to the mobile device and the cloud, however, they may include multiple cloud servers depending on the particular scenario that is being considered. There are other names used in literature for application offloading, for instance, *Application Outsourcing* and *Application Partitioning* are two of them ([32], [33], [34], [21], [35], [36], [37], [38]).

There are, in fact, multiple definitions of the problem depending on the particular scenario considered and the assumptions made. For example, the scenarios may include a single or multiple mobile devices, and a single or multiple servers. Furthermore, even when considering the same scenario, the goal of the problem may be different, e.g., minimizing the execution time for the mobile user or distributing the load among multiple servers. In this work, we focus on the application offloading problem for a single mobile device that offloads computations to a single cloud server, arguably the most common scenario addressed in literature. This assumption does not limit our work, however, as our contributions are on

issues that are common to other problem formulations such as modeling the application and its costs.

When we look at the big picture of the application offloading solution, we notice the interaction of several subsystems that perform different tasks. First, we have a subsystem that makes the offloading decisions, as discussed so far, which involves modeling the application, estimating its cost and finding an offloading decision. Estimating the application costs relies on information from the environment, such as the network bandwidth, that is typically gathered by profiling tasks. There are often three types of profilers, namely network, application, and mobile device profilers that gather, for example, information on the network's bandwidth, the costs of different parts of the application and the power consumption characteristics of the mobile device, respectively. In the scenarios where the mobile devices act as servers as well, a coordination mechanism to manage the Mobile Cloud is also required. Other subsystems may include a manager to start/shutdown virtual machines on the server, and managers that handle the offloading calls between the mobile device and the cloud server. The integration of these subsystems is the focus of some works such as that from Orsini et al. in [39]. The authors propose an integration framework called CloudAware, focusing on ad-hoc network and short-time interaction. The main components of this framework are a discovery service, a partitioner, a solver, a context manager and a coordinator. Our work focuses specifically on the offloading decision algorithm and the tasks it depends on, namely application and cost models, rather than focusing on the integration of the different subsystems.

We discuss some of the challenges of the application offloading problem, in particular, the execution, the application and the cost models, and the location decision algorithm.

2.2.1 Execution Model

The execution model describes how the state and execution of the application is transferred from the mobile device to the cloud. We classify the execution model in two main classes: *Client-Server* and *VM migration*.

In the *Client-Server* model, calls to components or modules of the application are offloaded from the mobile device to the cloud. The data required for the execution of the specific module call is transmitted using protocols with well established API.

These protocols may work at different layers of the OSI model. Typically, offloading schemes use network layer protocols such as RPC, RMI, Sockets, and REST protocol. However, some protocols at the network layer have been proposed recently to address

application offloading in scenarios such as C-RAN networks and MIMO antennas. Magurawalage et al. [40] propose a network layer protocol that uses an unified packet header and supports both task offloading and resources management in C-RAN and Mobile Cloud. Al-Shuwaili et al. [41] address the joint problem of application offloading and interference at uplink/downlink in MIMO systems, aiming at minimizing the energy across multiple mobile devices at multiple network cells. Similarly, [42] aims at minimizing the overall energy consumption by the mobile devices focusing on the network layer.

The Client-Server model requires the applications to be modified in order to use the offloading framework. Furthermore, the offloading service must be pre-installed in the remote server. These requirements, however, may not be realistic for some scenarios such as old applications and for some architectures such as Cloudlet and Mobile Cloud. Examples of client-server offloading are the method-based MAUI [2], OSGi-based solutions [43], Weblets [44], Code migration [12,45], and REST-based [46]. Client-Server offloading is known as Elastic Partitioned/Modularized Applications in [47].

Offloading through *VM migration* takes a different approach. The execution state of the mobile device, seen as a virtual machine, is transferred (migrated) to the server which is running a virtual machine in the cloud. The execution of the application is continued in the cloud-based virtual machine from the same point it was before migration. Hence, applications can be migrated to a server at any point of their execution, resume execution to run resource demanding processes, and then the new state is migrated back to the mobile device. The VM may be a snap-shot of a web application [45] that is migrated between the mobile device and the server. While there is no need to modify existing applications, VM migration requires transmission of considerably more data than client-server offloading.

Satyanarayanan et al. [9] discuss two approaches for data synchronization. On the one hand, the VM is suspended and its entire state is transferred to the Cloud where its execution is resumed ([48]). On the other hand, in the VM synthesis, only the difference between a previous image of the VM and the current state is transferred. Further, Jeong et al. [45] propose to not include common libraries used by the application. Regardless of the approach, Barbera et al. [49] show that short synchronization intervals lead to higher communication costs for offloading and data backup. Some VM migration-based examples are CloneCloud [50], [9] and [45]. VM migration is known as Augmented Execution in [47] as the application perceives an environment with augmented capabilities when running on the cloud. Regardless of which execution model we use, the offloading decision is tightly tied to the application model.

2.2.2 Application Model

The application model, also known as *program execution model* [51], describes the structure of the application, in particular, it specifies what parts or *modules* make the application and what the relations or *dependencies* between the modules are. The application model is at the core of the application offloading solution for two reasons. First, the definition of an offloading decision depends on what the modules of the application are and, second, the cost-benefit analysis requires not only the modules of the application, but also how their interact (i.e., their dependencies).

The level of abstraction that each module represents specifies the *granularity* of the offloading solution. In a method-based granularity, for example, each application module represents a method (sometimes also known as functions) in the application. Examples of granularity levels, ordered from finer to coarser, are bit-based [52], program instruction [50], method-based [2,12,53] and classes in Object Oriented Programming [10,54]. The finer the granularity, the more precise the application model and cost-benefit analysis are, however, it also increases the size and complexity of the problem. In contrast, coarse granularity results in less modules and simpler problem complexity but may result in suboptimal offloading decisions.

The *dependencies* between the modules describe the relationship between the execution of the modules, and are given by the several program instructions such as conditionals and parallel execution instructions. Some of the most common dependencies for a set of modules include the series dependency, where all the modules in the set run sequentially, the probabilistic dependency, where only one node in the set runs according to a probability function, and the parallel dependency where all the modules run in parallel. For example, in a method-based granularity, a method may invoke the sequential execution of two other methods. Additionally, nested execution of the methods leads to further complexity when estimating the total cost of the application. Naturally, the application model is the place to capture the information regarding the module dependencies. This information is key when performing the cost analysis of the entire application as we show in Section 2.2.3. However, the application models often simplify the types of the dependencies and/or the existence of nested executions. For example, a common simplification in application offloading works is to consider every dependency as a series dependency [50], [2], [53]. Another simplification is to consider that modules are self-contained, effectively excluding nested execution from the problem. Furthermore, cycle dependencies are sometimes modelled through a combination of the application and the cost models [2].

The use of application models is not exclusive to the application offloading problem however. Sahner and Trivedi present an application model in [55] where the application is modelled by series-parallel directed acyclic graph. The graph is transformed into a Decomposition Tree that is used for different goals such as application performance and reliability analysis. The work of Dennis [51] proposes a model for parallel program execution. However, instead of developing a model to perform application performance analysis, the model is developed to be used in the design of computer systems that are correct from the perspective of the application program execution.

We discuss the models used in application offloading next.

The Thin Client model

In Thin Client Model (TCM), the whole application is run in a surrogate server, and only a thin client application is run in the mobile device. The Thin Client application, running on the mobile device, collects user interactions and sends them to the server. In the server, all the computations are performed and the graphic output of the application is computed. The results are then sent to the Thin Client application which presents them to the user. In this model, there is no offloading decision to be made as everything is offloaded. The device requires a permanent connection to the server and the application cannot access the location services or device peripherals.

Thin client computing is appropriate for resource intensive applications or applications that cannot run in the device. In some cases, thin client computing is the only solution, for example when the mobile user wants to access a remote computer at work or at home. However, this approach is bandwidth intensive and requires an always-on connection. Furthermore, highly interactive applications like games might not be appropriate even under low RTT scenarios.

Thin client computing uses two types of protocols for client-server communication depending on the application requirements [56]. On the one hand, we have Citrix ICA [57], Microsoft RDP [58] and VNC [59] for slow motion applications like text editing or browsing. On the other hand, video streaming like standard H.264 is used for 3D games or watching videos. Hybrid solutions that dynamically choose the communication protocol are also possible. Typical examples of Thin Client applications are Microsoft Remote Desktop Connection [58] and Teamviewer [60], both allowing the user to log in a remote computer.

Star Graph Model

The Star Graph Model (SGM) is a simple application model that represents the application as a set of self-contained modules. A *self-contained* module does not depend on any other module, in other words, the execution of the module does not invoke the execution of another module. Also, modules may be independent. Two modules are *independent* when they do not share common data. Therefore, independent modules can run in parallel without requiring a synchronization mechanism. This is an important consideration for supporting parallel applications in application offloading in MCC. For its simplicity and flexibility, and because the model fits certain application types well such as embarrassingly parallel applications, this model is used often in literature [61], [62], [12], [24]. Figure 2.5 depicts an example of the SGM.

More formally, the application is represented by a graph $G = (V, E)$, where $V = \{v_0, v_1, ..., v_n\}$ are modules of the application. Module v_0 runs only in the mobile, and modules v_i may be offloaded, $0 < i \leq n$. Additionally, there is a link between v_0 and every v_i, $0 < i \leq n$. There are no other links, therefore $E = \{e_1, ..., e_n\}$, where $e_i = (v_0, v_i)$.

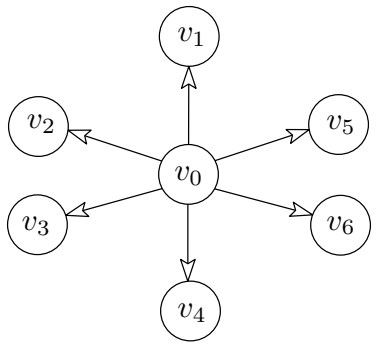


Figure 2.5: In Star Graph Model, each offloadable module is a node connected to the main node v_0.

Advantages This is a relative simple but precise model for applications composed by independent modules. Every time an module must be run, the offloading decision must be made considering the benefits of offloading only that module, thus decision may be relatively simple. This model fits well parallel applications and popular frameworks such as Hadoop [31], especially when the modules are independent.

Disadvantages The main limitation of this model is that module dependencies cannot be modelled and offloaded all in one single offloading operation. As a result, the SGM is

not appropriate for applications with a more complex structure, e.g., an image processing application that performs a group of image processing operations on an given image.

Call Graph Model

In the Call Graph Model (CGM), the application is represented as a directed graph where the nodes represent the modules of the application. The edges between two nodes express a direct relation between the modules. In contrast to the SGM, modules do not need to be self-contained and thus may depend on other modules. In some works, the CGM is restricted to directed acyclic graphs.

Formally, an application is represented by a graph $G = (V, E)$ where the nodes in $V = \{v_0, ..., v_n\}$ are modules of the application. There is an edge $e_{ij} = (v_i, v_j)$ between two modules v_i and v_j if module v_i depends on module v_j. The module represented by v_0 must run on the mobile device. The offloading operation may happen at any edge $e_{ij} = (v_i, v_j)$, in which case both modules would be run in different contexts. In some studies [2], [53], if a module is offloaded, all the modules invoked from said module will be offloaded as well. Furthermore, some works use a Call Tree which is effectively a CGM where modules are invoked from only one other module [50], [53]. Figure 2.6 illustrates an example of a CGM.

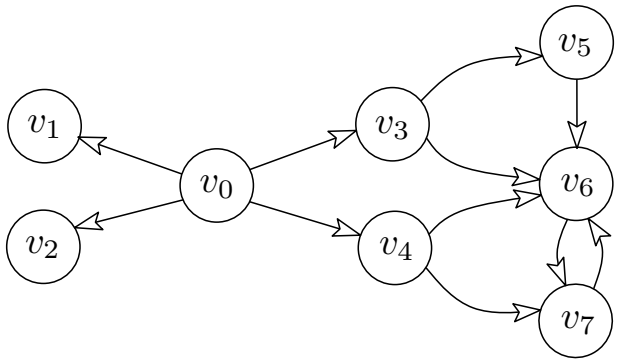


Figure 2.6: Example of the Call Graph Model.

Advantages This model supports applications with more complex structures compared to the SGM. As a consequence, the model allows offloading of a set of modules in one single offloading operation. For example, offloading v_4 in Figure 2.6 will offload the calls from v_4 to modules v_6 and v_7. Furthermore, the CGM supports more complex cost analysis as the cost of running a module may include the cost of dependent modules.

Disadvantages The CGM only models the sequential module dependency. Other common programming instructions such as conditionals, cycles and parallel applications are not supported. This limitation affects the accuracy of the cost estimation and may result in suboptimal offloading decisions.

Workflow Graph Model

The Workflow Graph Model (WGM) follows the workflow programming paradigm and models applications that can be modelled as workflows. In workflow models, the application data experiences a set of operations until the data is delivered in a final form(s). As a result, WGM is well suited and restricted to applications such as image processing or voice-to-text, or workflows with a defined set of operations. Figure 2.7 presents an example of a WGM for an application with six operations. In the workflow terminology, the operations are called activities. However, to be consistent throughout this work, we will use the term *module* as used above.

The application is modelled as a graph $G = (V, E)$ where every vertex is a module. The modules are related with each other according to different workflow patterns. Example of these patterns can be the *Sequence*, *Parallel Split* and *Choice* patterns, where two modules must run in series, in parallel or only one module runs with a given probability, respectively. Van et al. address the workflow patterns in [63]. These patterns between modules are modelled by edges in the graph. A main difference when compared to the SGM and CGM is that module dependencies are reversed, i.e., the execution of parent activities complete before the execution of the children. Also, the modules are self-contained as in the SGM. As this model is restricted to workflow-like application types, it is used less often in literature [64], [65].

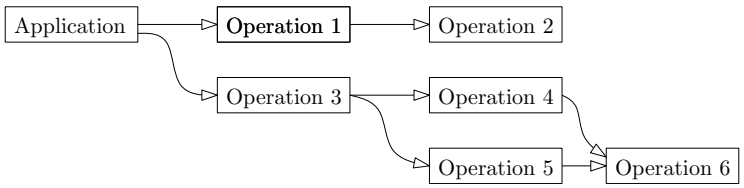


Figure 2.7: Workflow Graph Model as a workflow model for a six-module (activity) application.

Advantages The WGM supports module dependencies according to the workflow patterns. As a result, it may capture the structure of the application accurately, even though in a data-centric view. This is appropriate for data-centric applications.

Disadvantages However, the data-centric view limits the types of applications that can be represented by the WGM. Furthermore, the model requires knowing all the possible workflows of the application, which may not be practical for large applications. Also, the modules must be self-contained.

We have described four models, i.e., TCM, SGM, CGM and WGM, and mentioned the application types they suit best. We have seen that applications made of self-contained modules may be modelled by the simple SGM. The CGM supports series module dependencies for general applications, while the WGM supports other module dependencies for data-centric applications. Table 2.1 summarizes the support of some application characteristics by each application model.

Mobile Application Characteristic	TCM	SGM	CGM	WGM
Single module	✓	✓	✓	✓
Series dependencies		✓	✓	✓
Probabilistic and parallel dependencies				✓
Nested executions			✓	✓

Table 2.1: Application model support per mobile application characteristic. While the Workflow Graph Model considers more module dependencies, its use is restricted to data-centric applications.

In addition to defining the application model, solving the application offloading problem involves performing a cost-benefit analysis.

2.2.3 Cost Model

The cost model is used to estimate the cost of running the application, and thus it is essential to define an optimization function for the offloading problem. The estimate is computed by modeling the costs of each module first and then aggregating these costs according to the dependencies between the modules. Therefore, the main tasks are (1) modeling the cost of the modules, and (2) estimating the cost of the application based on their dependencies. The application cost estimate is later used as the objective function (and possible constraints) of the optimization problem, and thus it must reflect the ultimate goal of the offloading problem.

One of the goals of the cost model is to evaluate the cost tradeoff. While offloading saves resources by performing computations on the cloud, the process of offloading comes with its own costs due to, for example, the data transmission required to perform the migration. This tradeoff is analyzed in [1] by Kumar and Lu and can be summarized in Figure 2.8.

This figure expresses that offloading is beneficial when the module have high computational processing cost and offloading requires low data transmission. On the contrary, when the amount of data to be sent is high and CPU processing is low, offloading has a negative effect on the battery of the device and performance of the application. For the rest of the cases, i.e., when the amount of data to be transferred is in the same order as the required computations, the best offloading decision depends on the available bandwidth. In [66], Ahmed et al. recommend application designers to minimize the size of the application running states to reduce the impact of the network in offloading.

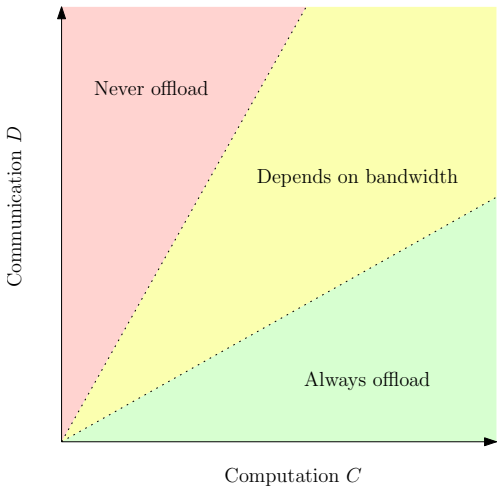


Figure 2.8: Tradeoff between computation and communication costs of an execution module [1].

Modeling Environment Conditions

Several variables from the environment affect the performance of the application. Some examples include the device energy rate consumption, the CPU clock speed, the number of hops from the mobile device to the cloud and the usage pattern of the mobile user. For example, Abolfazli et al. [32] show that the network delay depends on the number of hops between the mobile device and the cloud rather than on the physical distance between them, while Ahmed et al. [66] show that a higher number of users on the network increases migration time.

The cost model must take into account these variables or make simplifications and assumptions. For example, in [67], the energy cost of the application is estimated considering the clock frequency of the CPU, which is changed by modifying the voltage of the CPU. In MAUI [2], the energy consumption of a module is estimated as the number of CPU

instructions multiplied by the average energy cost per instruction. However, it is common to make simplifications that capture the impact of multiple variables within one single variable.. For example, all the variables concerning the network, such as the number of hops between the mobile and the server, may be included in a variable that estimates the network bandwidth. The abstraction is also done, for example, to model the time cost of the modules. Said time cost implicitly include the CPU clock speed and the number of instructions of the module, and it is estimated by profiling [50, 68].

The estimates of the parameters that affect the application cost are often gathered by *resource monitoring and profiling* [2], [24]. In *application profiling*, the execution of the application is monitored and profiled, gathering information such as energy and time consumed for subroutines of the code, and data to be transferred. Subsequent executions of the code are profiled to update such information. *Device profiling* deals with different metrics such as energy per CPU cycle or per KB sent over WLAN. Finally, *network profiling* measures the network status such as bandwidth, which is used to estimate the transmission cost of the modules. Profiling periodically keeps the estimates more accurate upon changes and it has shown accurate [68], but comes at an additional cost.

Module Costs

The costs of each module are the building blocks of the cost model. They are the smallest cost units that constitute the cost of the entire application. Typically, we have three costs per module, namely, execution cost in the mobile and in the cloud, and the transmission cost between both locations when the module is offloaded. The following characterization of these costs applies to all of these three cases.

There are multiple variables that affect the module costs, such as the state of the application and of the network at the moment of execution of the module, that results in different costs for different executions. Therefore, each cost can be considered as a random variable with a distribution function. For example, Figure 2.9 depicts a call graph of an application and the histogram of the execution cost in the mobile in seconds for module d.

As we mentioned, there are multiple variables that affect the module's cost. It would be desirable to include as many of these variables in the model in order to increase the accuracy of the model. Among the variables, we have the state of the application, which includes the execution path of the call to the module. The *execution path* of a module's execution is the sequence of module calls leading to the execution of the module, which is equivalent to the state of the call stack at the time of executing the module. For example,

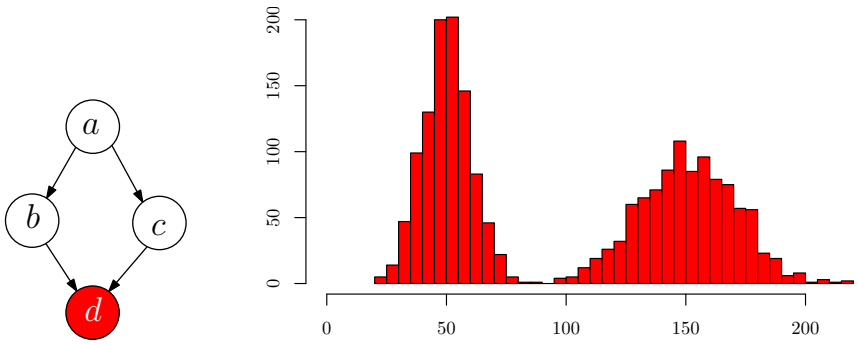


Figure 2.9: Histogram of a module's cost.

module d has two execution paths in Figure 2.10, i.e., abd and acd. When the execution path is abd, then the module a was invoked first, which in turn invoked module b, which finally invoked module d. Similarly, path acd means that the call sequence is a, c and then d.

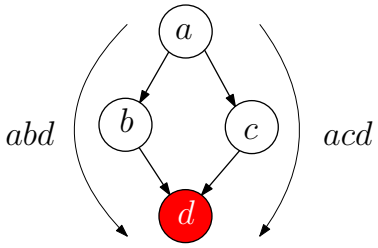
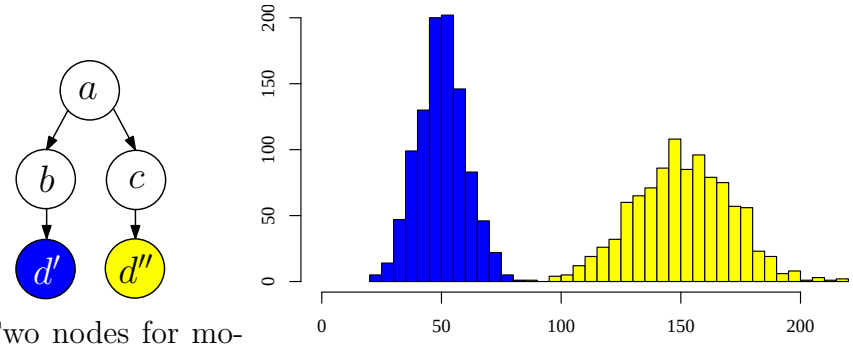


Figure 2.10: Two execution paths to module d.

As the costs of module d may be different according to the execution path, it is more accurate to represent the costs considering the execution path. In fact, this information can be included in the application model using multiple nodes per module as shown in Figure 2.11a, which shows two nodes in the graph for module d, namely d' and d''. Consequently, the cost for module d is captured separately for each execution path by nodes d' and d'', even when they represent the same module in the application. Figure 2.11b depicts a the histograms for each node separately.

Application Cost

The goal of the offloading problem is to minimize the application cost and, therefore, we must compute an estimate of it. The cost of the application refers to the total execution cost, for example, time or energy consumed. Some formulations of the problem aim at minimizing the energy consumed while keeping the execution time under a threshold [2]. Furthermore, the estimate must account for the offloading decision for each module. Some



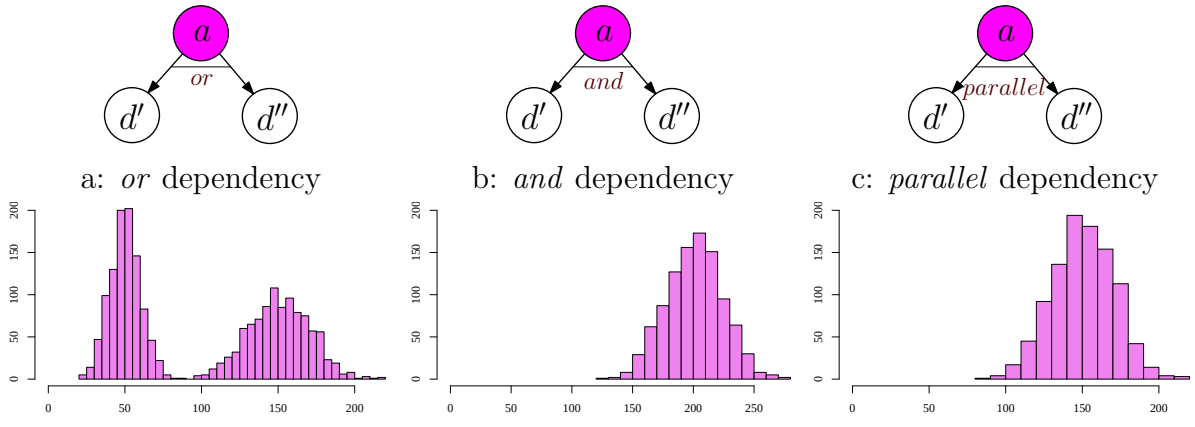
a: Two nodes for module d.

b: Histograms of costs of nodes d' and d''.

Figure 2.11: Multiple nodes and cost histograms according to the execution paths for the same application module.

works estimate the application cost as a sum of the cost of each individual module based on whether the module is offloaded or not. However, the execution cost of the entire application must be computed based not only on the cost of the individual modules, but also based on their dependencies. We should illustrate the impact that these dependencies have on the application total cost.

In order to understand how the dependencies between the modules affect the cost of the application, we use the example depicted in Figure 2.12. The figure illustrates the call graphs and the histograms of the cost of three applications that differ only on the dependencies of the modules. The three applications consist in a node a that calls two other nodes, d' and d'', using conditional (or), sequential (and) and *parallel* instructions.



a: *or* dependency

b: *and* dependency

c: *parallel* dependency

d: Histogram *or*

e: Histogram *and*

f: Histogram *parallel*

Figure 2.12: Module dependencies, i.e., conditional (or), sequential (and) and *parallel*, and the histograms of the application cost.

For instance, the *or* dependency, Figure 2.12a, refers to a conditional dependency, i.e.,

module a invokes either module d' or module d''. One execution the application is then the execution of a and then the execution of *either* d' or d'', and thus the cost of the application is the cost of d' or of d'', depending on which module was executed. Let us consider that the cost of a is negligible, and that d' and d'' have the same costs as in Figure 2.11. For clarity in the presentation, the example only considers execution cost and no offloading decisions. Assuming equal probability of running each of d' and d'', the histogram of the running cost of the application is depicted in Figure 2.12d. Similarly, the call graph in Figure 2.12b models the application using the sequential dependency (*and*), where Figure 2.12e shows the histogram of the application. Finally, the application's call graph for the *parallel* dependency and the histogram of the costs are shown in Figures 2.12c and 2.12f respectively.

We can see that the distribution of the application cost is different for each dependency. For instance, for the series dependency, the application mostly takes either between 25 s and 75 s or between 100 s and 200 s. However, for the conditional dependency, the application cost is mostly between 150 s and 250 s while the parallel version takes between 100 s and 200 s to finish. This example shows that the modules' dependencies must be considered in order to obtain an accurate estimate of the application cost.

Uncertainty

In some scenarios, we have to deal with incomplete cost information that prevents us from building a distribution function of the module's costs. This occurs, for example, when we have not yet profiled enough data to build an accurate cost distribution function. Other times, we may have the costs' distribution functions but working with them is computational expensive and may not be an option, for example, when running the offloading method on the mobile device. Therefore, we often have to simplify our cost model. We may use the average cost or other relevant information to define feasibility and optimality of the solutions to the offloading problem.

Typically, the cost is modelled by a single value that represents the expected cost, e.g., [2]. For example, the execution time in the mobile of a module is represented by the average time of all the executions measured in the mobile for the module. In the example of Figure 2.9, the average execution cost of module d is about 100 s. Another approach is used in Scavenger [12] where the application developer must specify the execution cost of each offloadable method (module) as a function of the parameters of the method call. This idea, however, transfers the responsibility to the developer, and it is not practical as

the function must also consider external factors such as the hardware the application is running on.

The cost model and application model are the main abstractions that support the design of the offloading decision algorithm.

2.2.4 Decision Algorithm

The decision algorithm must find an offloading decision for the application's modules such that the execution cost of the entire application is minimized.

Solving the application offloading problem requires defining an *objective function* which to be used to decide the best running location for each module. Other considerations to be considered are the preference of the user and the state of device, e.g., if the mobile device is connected to a power source, the user would rather minimize the execution time of the application instead of the energy consumption. The application offloading solution may also aim at fairness and optimizing resources shared by multiple users such as shared network bandwidth in a Mobile Cloud. In [69], Liang et al. propose a model that performs resource allocation for security services considering different security requirements and costs, aiming at maximizing the benefits of the cloud operator under constraints that are based on user requirements.

In addition, the problem must consider *constraints* such as specifying a cap in the monetary costs of using the cloud, or a maximum running time when minimizing the energy consumption. Also, the modules may have context restrictions, for instance, some modules may be able to run only on the mobile device, only on the cloud, or both. As a result, the offloading solution must carefully specify its optimization function and constraints. Even more, the solution should support different optimization goals and constraints depending on user preferences and environment conditions.

Algorithm For SGM and CGM, it is common to see variants of Linear Programming (LP) formulations [2], [70], [50], [71]. Another approach for SGM is to perform offloading decisions as a scheduling algorithm that decides the running location of the module call when the call is issued [68]. A similar approach is presented by works based on frameworks such as OSGi [72], [73], [43] and Hadoop [74], [24]. In such cases, the offloading decision is implicitly done by the framework's load balancing, scheduling, and fault tolerant algorithms.

Static/Dynamic Static/Dynamic refers to the moment(s) the offloading algorithm is run. However, the distinction is not always clear. Some static algorithms are run before applications execution, producing a set of possible solutions for different parameter ranges. Then, during application launch, the most appropriate solution is chosen according to the state of the environment at that time [50]. Other approaches run the algorithm upon application execution, and the offloading decision is recomputed in response to environment changes such as network connectivity. This approach is closer to be dynamic [2], [75]. Even more, dynamic approaches follow a scheduling-like approach, and the decision is computed at the time each module must be run, considering factors such as server and mobile device capacity [68]. Close to this approach are those based on frameworks such as Hadoop and OSGi which run tasks/jobs in the context according to load balancing and fault tolerant algorithms [72], [73], [43], [74], [24].

Mobile or Cloud Execution Running the offloading decision algorithm incurs a cost. Therefore, a key decision is where to run the algorithm itself, either on the mobile [2], [68], or on the cloud [50]. On the one hand, if run on the mobile device, a more accurate picture of the environment is available and the offloading decision is more accurate. However, the device resources are used to run the algorithm itself and this cost could negate the benefit of offloading. Furthermore, in order to cope with the limited computing power of the mobile device, the algorithm might require problem relaxations resulting in sub-optimal solutions. On the other hand, running the algorithm on the cloud allows for a complex and precise problem formulation and algorithm. However, the offloading solutions must be communicated to the mobile, a process that consumes energy and time. Even more, environment conditions might have changed by the time the mobile receives and implements the offloading decision, resulting in an outdated and suboptimal solution.

2.3 Existing Approaches

Solutions for the application offloading problem differ in multiple aspects such as the MCC architecture (e.g., Mobile Cloud), the optimization goal (e.g., application performance and resource fairness in Mobile Cloud), the execution model (VM-migration or client-server), the communication protocol (e.g., RPC and REST), the algorithm design technique (e.g., LP and heuristics) and the metrics they consider (e.g., network data rate and location aware). We now describe the most relevant to our work grouping them according to the application model used.

2.3.1 Star Graph Model

In this section, we present several solutions that use the Star Graph Model and thus consider the application as a set of independent tasks.

Balancing Performance, Energy and Quality in Pervasive Computing (Spectra) Flinn et al. present Spectra in [61]. In Spectra, an application is seen as a one module that runs in the mobile, and a set of self-contained modules that can be offloaded. Therefore, Spectra's application model is the SGM. The developers of the application must specify the application modules that can be offloaded. Furthermore, the application must specify the possible offloading solutions to be considered by the offloading framework. Then, the framework evaluates the execution cost of each solution and chooses the solution that provides the best application performance.

Spectra leverages Odyssey [76] which includes the concept of fidelity. *Fidelity* refers to the level of precision used in the operation, providing another degree of flexibility to lower the running costs of the application. For example, the fidelity levels may be different image resolutions and frames per second in video streaming. Thus, the Spectra framework notifies the application with the offloading decision and the fidelity level according to the rules specified by the application and based on the status of the environment (status of the mobile, the communication network, and cloud server). While adding another degree of flexibility, fidelity is not applicable to all application domains, for instance to exact mathematical computations.

The framework performs profiling of different resources, from CPU and memory usage in the mobile and in the remote servers, up to resource availability. It also uses a model to predict resource usage of the modules. After finding the servers that could accommodate the module according to its predicted resource needs, the framework makes the offloading and fidelity level decisions by maximizing an utility function. The goals of the function is to improve application performance from the user perspective. Therefore, it includes energy consumption, time, and fidelity. The actual calculations of the default utility function are not presented in the work.

Tactics-Based Remote Execution for Mobile Computing In [62], Balan et al. present *Chroma*, a solution where offloading modules are self-contained and where offloading happens via RPC-calls. Offloading location decisions are called *Tactics*. Beside the location of the modules, Tactics also include the *fidelity* similar to [61] and presented in

Odyssey [76]. Applications are made of independent and self-contained operations, therefore the Star Model. As required by RPC-calls, application code must be present in both the client and the server. Chroma uses a continuous monitoring system to adapt to changing environment. Applications can have several possible Tactics. Tactics are defined by the developer and define the amount of resources used and the fidelity. The system then chooses a tactic based on the state of the environment at runtime.

Scavenger: Transparent Development of Efficient Cyber Foraging Applications Scavenger in a framework for offloading (called outsourcing in the work) presented in [12] by Kristensen. It tackles all the main aspects of offloading, i.e., application and device profiling, offloading points decisions, and provides a model that abstracts the programmer from the implementation details to make the application offloading-enabled. Scavenger relies on RPC for client-server communication. It provides two offloading mechanisms: an automatic approach where all the offloading is automatic, and a manual scheme where the application must get the server address and make the calls explicitly. Also, the programmer is not totally agnostic of offloading, like in MAUI [2], it must provide the code with annotations of what methods should be considered for offloading. It also includes notions of security such as black and white lists for validating client devices.

In Scavenger, offloading is performed by sending the code of the offloaded functions. Developers communicate to the scheduler which functions are candidate to be offloaded and their complexity based on the input parameters. They can also specify the size of the input and output of the functions. This information is used by the scheduler when deciding if the method should be offloaded or not. Offloadable functions must be self-contained.

Virtual Cloud Huerta et al. [24] present an architecture and solution for Mobile Cloud. They address the scenario where devices work as clients and also as surrogates servers. In their model, devices interconnect forming a cloud, and are classified in stable devices or not. A device is stable when it is stationary or its trajectory can be predicted. Only stable devices are considered for serving as servers.

Their solution is Java-based, and code injection is performed to the applications upon start. The code injection specifies the migration points. However, there is no mention of a benefit function, granularity, etc. Offloadable units are called Jobs. The applications consist of a main program and jobs that can be offloaded, being the jobs independent and self-contained. Therefore, they are implicitly using a SGM. They use a distributed file system to share files within the cloud, while RPC is used for remote execution and

XMPP [77] for cloud management. There is no mention of a cost evaluation and decision algorithm for offloading or content synchronization before and after offloading.

Grid design for mobile thin client computing In [56], Deboosere et al. focus on a grid design for thin client computing. The authors state that RTT between the client and the server is a major factor affecting the user experience as it introduces delay, mentioning that RTT is even more important than bandwidth. Therefore, appropriate location of the server is crucial. Authors propose an algorithm to locate new servers within a grid network and to choose them according to the location of the client. Also, as clients are mobile, migration of the application between servers will be required to keep running the application in a server with a low RTT to the new locations of the client.

Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones In [67], Wen et al. discuss a model to optimize mobile energy consumption by offloading applications to cloud servers using cloud clones (VMs). Applications are completely run in either the server or in the mobile device, similar to a thin client scenario. As a result, the offloading method must solve two optimization problems to find the energy required in each case. In the first case, when the application runs in the mobile, the authors propose to control voltage provided to the CPU in order to save energy. However, the CPU frequency is linear with respect to voltage and so is the CPU's number of operations per second. As a result, the time required to run the application will vary with the varying CPU voltage. The solution aims at finding the best assignment of voltages during the application execution that minimizes the energy consumed to run the application while not exceeding a predefined execution time. A statistical analysis is performed to estimate the energy consumption rate and speed of the CPU for different CPU voltage values. This estimated values are used to define an optimization function to find the CPU voltage that results in minimum energy consumption. In the second case, when running on the server, energy is required to transfer required data from the mobile device to the server. The amount of energy required depends on the amount of data to be sent and on the current state of the network. Having the required energy for both scenarios allows to choose the one that saves more energy for the mobile device.

Multiprocessors Message Passing Interface (MMPI) a message passing interface for the mobile environment Doolan et al. [25] propose a framework for running highly parallel computing problems on a Mobile Cloud. This work leverages Java's Message

Passing Interface for communication over Bluetooth similar to RPC calls. The application is composed of self-contained operations.

CloneCloud: Elastic Execution between Mobile Device and Cloud In [50], Chun et al. propose CloneCloud. CloneCloud performs offloading by migrating application execution threads to virtual machines in the cloud. The offloading decision of the application is decided offline, i.e., before the execution of the application. Several offloading decisions are computed for different environment conditions such as network datarate. Upon execution of the application, a offloading decision will be chosen according to the current conditions; the executable code is modified to describe such solution.

Offloading may happen at any point of the application providing fine granularity. However, there must be no nested offloading operations: once a thread is offloaded to the cloud, there may not be another migration instruction in it. Therefore, modules are self-contained.

As offloading happens at a thread level, it is also important to synchronize data shared between different threads. In this case, if a thread accesses a resource that has been migrated, such thread blocks until the migrated thread completes and the state of the application is merged back to the mobile. However, the offloading problem is solved offline, therefore once the application starts running, its partitioning and offloading decisions do not change. This may result in bad application execution performance if environment change.

CloudClone is a system that allows offloading of applications by migrating the application to a virtual machine. It uses a combination of static and dynamic profiling in order to determine points for offloading. It abstracts the programmer of the process, and it does not require modifying the applications to make them offloadable.

The static portion of profiling detects constraints in the migration points, while the dynamic portion builds a cost model used to determine when to offload or not. An optimizer selects the migration points that optimize the objective (energy consumption or execution time for example). In addition, the system works at a thread-level. This allows threads that are not device constrained (threads that do not use the mobile peripherals for example) to run and be offloaded independently of other device constrained threads.

Sharing-aware Cloud-based Mobile Outsourcing [21] Mei et al. [21] address the problem of improving performance by sharing data across multiple applications. Authors present the design of a mobile application outsourcing framework and its implementation

for Android and Amazon EC2. They use the cloud for data sharing after the offloading decision have been already made. The developed algorithms use data sharing between applications in order to improve outsourcing performance, i.e., lower network overhead and lower runtime. They follow a module-partitioning view of the application and try to collocate fine-grained modules based on data sharing in the cloud. The multi-application data sharing is done by using data mining techniques to detect where it can be done. Their proposal is a client-server model with no state even though some cache is possible. Additionally, by using the capabilities of the Cloud, the framework is scalable in the number of mobile users. The article's focus is on the platform's mechanisms for offloading, managing information about of available servers, assigning them to the clients, and the scheduling of offloading requests. The article does not discuss an offloading decision algorithm.

MARS: Adaptive Remote Execution for Multi-threaded Mobile Devices MARS (Multi-threaded Adaptive Remote execution Scheduler) [68] is a client-server RPC-based remote execution scheduler. The offloadable RPCs are segments of the program specified by the developer and must be self-contained and independent which results in the Star Graph application model. The cost of the RPC calls is estimated via a ranking metric that estimates the gain in time of performing offloading for each RPC. The metric considers local and remote execution time, upload and download size of the RPC call, and upload and download network bandwidth. In addition, the framework defines a ranking metric for energy saving that also includes the device CPU power and upload and download device power consumption. To minimize the application's execution time, the framework defines a heuristic algorithm.

The heuristic scheduling algorithm keeps the list of RPCs that must be executed which are ranked according to their performance ranking. Then, when a remote resource is available, the method with higher ranking is offloaded to the remote server. Similarly, when a core in the mobile device becomes free, the RPC with the lowest gain is chosen and run in said core. In addition, the algorithm only offload RPCs with an energy gain estimate above a threshold to prevent high energy costs. To adapt to changes in the environment, the framework updates both metrics and sorts the list of candidate RPCs every time a new method needs to be scheduled. Similar to other operating system scheduling algorithms, starvation must be prevented. This is achieved by defining a sets of bins where each bin holds a group of RPC calls that have similar offloading ranks. The authors performed simulations for three different CPU intensive applications: face recognition, augmented reality, and video game. Their simulations included three different network conditions, i.e., outdoor WLAN, indoor WLAN and 3G, and two different mobile devices and an

Amazon EC2 server. The results show an improvement in time and energy efficiency for the three applications for WLAN scenarios and not for 3G.

This scheduling heuristic is light enough so it can run on the device while it also adapts to changing conditions. The authors claim this method outperforms best static client-server offloading solutions. Furthermore, it supports systems with multiple threads and cores.

Cuckoo: A Computation Offloading Framework for Smartphones Kemp et al. present an offloading solution named Cuckoo [78]. The offloadable units are Android services which are computations that can be run in different threads and do not access the interfaces of the mobile device. The proposed model is the Star Model where the application is made by self-contained services that can be offloaded. Offloading is performed by sending the source code as part of the offloading operation. This increases the transmission costs and may raise security concerns, however, the mobile application does not need to be deployed and synchronized with the server.

ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading In [11], Kosta et al. propose a framework for self-contained method-based mobile application offloading. This solution takes elements from both CloneCloud [50] and MAUI [2] being a VM-based offloading solution with a method-based granularity. Method executions are offloaded to virtual machines running the mobile system. Parallel computations can be offloaded to multiple VMs simultaneously. The application can request resources to the cloud so new VMs are created on request. Every time a method runs, its costs are measured. The decision of offloading is made when the method is to be executed based on previous execution measurements. Such decision is made according to a specified policy, either to minimize time, energy, both, or a combination of these and monetary cost. Simulations are performed to demonstrate resource savings with the framework, also with parallel executions using multiple servers, and finally the ability of request VMs with increased resources. ThinkAir can offload parallel processes. However, the network bandwidth is shared among simultaneous offloading operations which may increase the cost of offloading simultaneous modules.

Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices Shi et al. [26] aims at improving the performance of mobile applications leveraging the Mobile Cloud. The goal is to minimize the application execution cost in the

mobile device, i.e., save energy and time, by using resources of surrogate mobile devices, and under different conditions such as intermittent availability. The authors consider two subproblems, firstly, how to model the computational job and divide it into tasks, and secondly, the problem of how to distribute said tasks for remote execution in mobile clouds. Jobs are made of set of tasks and thus this model is equivalent to our Star Graph Model where tasks are self-contained offloadable units. In the proposed architecture, tasks are disseminated across devices by a job engine. A master process controls resources in each surrogate mobile device and communicates with the job engine. For each task, a profile of the task resource needs is kept. This information is then used along with a job priority by the job engine in order to distribute tasks among the surrogate mobile devices. A Serendipity implementation for Android was developed and experiments where run, showing the feasibility of the approach.

Cloud Server Job Selection and Scheduling in Mobile Computation Offloading In [79], Yue et al. address the problem of multiple mobile devices offloading self-contained jobs to the cloud. The authors address scheduling of the jobs under the constrain of a shared network and propose an offline and three online scheduling algorithms. The algorithms aim at minimizing the energy used by the mobile devices when accessing the base station. The mobile devices use a common wireless base station to access the cloud, and the server instructs the devices when and what job to offload and what job to process locally. The server uses the offline scheduling algorithms to schedule the jobs generated by the mobile devices. The online scheduling algorithms aim at improving fairness at the server for all the mobile devices. To schedule the communication to the base station, time is divided in discrete intervals which are then assigned to jobs per mobile so as to minimize the energy used by the mobiles.

The problem addressed by Yue et al. [79] can be extended to consider multithreaded applications in a single mobile device. These problems are equivalent. For instance, each thread can be seen as one mobile device in the multi-device problem. Scheduling access to the base station is equivalent to scheduling the use of the network in the single mobile device with multi-threaded applications.

Online Algorithms for Location-Aware Task Offloading in Two-Tiered Mobile Cloud Environments Xia et al. [80] study a two-tiered cloud architecture where mobile devices access a Cloudlet via multiple WLAN access points, and distant clouds via the cellular network, and where the offloading decision is done according to a location-aware

offloading algorithm. The optimization goal is to balance the energy cost for each mobile device depending on the energy available on the device while meeting the software level agreements. Mobile devices offload tasks and the framework evaluates the cost of the task considering the required and remaining energy on the mobile and following a time division scheme. The cost model considers the device residual energy. The application model is the Star Graph Model as tasks are self-contained.

Advancing the State of Mobile Cloud Computing A REST-based offloading scheme is proposed by Bahl et al. in [46]. This is a stateless model where functionality common to different applications can be provided through REST services. Examples of such functionality are face detection or speech recognition operations. The application model is a star dependency graph where the application is the center of the graph and there is a dependency to the services. Only the services can be offloaded.

The advantage with REST-based offloading is simplicity and minimum effort as apps can start using functionality already provided and working. Also, REST protocols are widely used and stable even though they might not be the best option for data communication performance. Finally, REST services are platform independent, thus virtually any mobile application can use these services. The services may be developed by third party companies, however, this arises concerns such as compatibility, availability and security. Compatibility is a concern for the application when the service is updated possibly changing its interface and thus forcing application updates. Availability problems arise when the mobile device goes offline and then the application has not a local implementation of the service functionality and thus must fail. Furthermore, the service may be available freely on internet to possibly millions of users competing for its resources. Finally, security is a concern as the REST-services may be developed by third party companies that will have access to the user's information contained within the service call, and thus privacy of user's data is at risk.

On Energy-Efficient Offloading in Mobile Cloud for Real-Time Video Applications In [81], Zhang et al. propose a scheduling algorithm as an adaptive dynamic offloading solution. Authors focus on real-time video applications, for example, games where the scenario is rendered as a video. The offloadable unit is a task, which are independent and therefore the application model is the SGM. Each task has associated energy costs, i.e., computation cost and amount of data to be sent and received, as well as associated start time and deadline. The algorithm follows a heuristic greedy approach aiming at

minimizing the energy consumption subject to deadline constraints. Authors also propose a network state monitoring.

Energy and Delay Tradeoff for Application Offloading in Mobile Cloud Computing Wang et al. [82] address the problem of minimizing both the energy consumption and execution time of multiple applications from multiple mobile devices, and by offloading the applications to multiple servers. The applications are independent, resulting in a SGM application model. They prove that the problem is NP-hard for the general case and propose a heuristic algorithm. The problem is presented as a 0-1 optimization problem, where variable $x_{m,a,s} = 1$ if application a of the mobile device m is running on server s. The optimization goal is to minimize the sum of the energy and time costs, which might not be appropriate as both time and energy are in different units. To address this issue, the authors multiply the energy cost by a constant thus adjusting the preference to optimize between time or energy costs. However, it is not clear how this parameter should be assigned. The algorithms are evaluated on randomly generated applications and costs.

Optimization of Radio and Computational Resources for Energy Efficiency in Latency-Constrained Application Offloading Munoz et al. [52] address the scenario where both the mobile device and the access point use MIMO antennas, and analyze the tradeoff between energy consumption and latency. The work proposes a framework for the joint optimization of the radio and computational resource usage. The goal is to minimize the energy consumption while keeping the QoS as measured by a latency (deadline) constraint that depends on the user's perception. The granularity of the offloading method is the bit level, i.e., the application is considered as a group of bits that can be offloaded or not. The method chooses how many bits to offload and how many to 'run' on the mobile device. The time and energy costs are estimated from the number of bits, which are considered independent, i.e., the cost of 'executing' a bit does not affect the cost of other bits, which corresponds to the SGM application model.

Make Smartphones Last A Day: Pre-processing Based Computer Vision Application Offloading In [83], Li et al. take a different approach to save energy by offloading. They authors propose to sacrifice accuracy in order to satisfy energy and time constraints. For instance, for a face detection application, preprocessing the image (downscaling it) results in lower transmission cost to the server. The decision algorithm is inspired in Lyapunov optimization. The authors propose a scheduling algorithm for migration of

multiple independent tasks. Preprocessing introduces a new tradeoff, it has the potential save energy and time when offloading, but it also consumes energy and time to perform the preprocessing task.

Offloading of Web Application Computations: A Snapshot-Based Approach Jeong and Moon [45] use offloading in Web applications. Offloading uses snapshot of the state of the web application which allows the server to use Javascript handlers and DOM API. Information from common JavaScript libraries is omitted to reduce the amount of data to be transferred. The snapshot approach is similar to VM-migration, where the state of the application is sent to the server. The scheme requires blocking the application at the client. The scope of the work does not include a decision algorithm.

A Mobile Application Offloading Algorithm for Mobile Cloud Computing In [84], Ellouze et al. propose an offloading algorithm with focus on energy efficiency and quality of experience (QoE). The algorithm works a the level of operating system and offloads independent jobs, each one corresponding to an application. A job is offloaded if it reduces the energy consumption while the execution time remains within a time frame. The method considers the QoE in terms of additional delay of the jobs. They incorporate models of wireless data rates, i.e., signal-to-noise ratio, bandwidth, distance, etc., to the model to compute the expected time and energy during transmission. In the simulations, the authors experimented with different job arrival rates, and three types of jobs, namely, chess application, voice recognition, and virus scanning. The solution proposes a CPU sharing scheme as part of the algorithm, where the CPU run parts of each concurrent job.

Ellouze and Gagnaire [85], extend this work [84] to the scenario where servers can be located at multiple sites. They first provide an analytical analysis of the performance of the approach [84] based on the network bandwidth and the capacity of the servers. Finally, they propose an optimized application servers placement in a typical C-RAN configuration.

2.3.2 Call Graph Model

An Adaptive Multi-Constraint Partitioning Algorithm for Offloading in Pervasive Systems An example of modularized applications is presented by Ou et al. in [10]. The authors represent the application as an undirected graph where nodes are program classes of an object oriented program, and edges express dependency between classes. The algorithm finds a $k+1$-partitioning of the graph which is used to decide the offloading

solution, reducing the search space. One partition includes all the classes that must run in the mobile device whereas the other k partitions can be offloaded. Nodes and edges are multi-weighted corresponding to memory, CPU, and bandwidth requirements, and the number of times classes access each other. This is probably the base for application model of many later works based on the Call Graph model.

MAUI: Making Smartphones Last Longer with Code Offload Cuervo et al. present MAUI in [2], a client-server architecture to preserve energy in mobiles devices by offloading computations to cloud servers via RPC calls. The framework is method-based and developers must annotate which methods should be considered for offloading. The application is modelled via a call graph where methods are represented by nodes in the graph, and where there is an edge between two methods if one method may call the other. To estimate the cost, nodes and edges have weights representing the computation and communication costs respectively. The offloading problem is formulated as an integer linear programming problem where the objective function is to minimize the energy cost of running the application. The framework gathers profile information during execution of the application and uses said information in future decisions. Furthermore, Maui continuously measures of the current communication link state such as the bandwidth and latency of the link.

Computing in Cirrus Clouds: The Challenge of Intermittent Connectivity Shi et al. address MCC under intermittent connectivity which they name Cirrus Clouds in [3]. The authors describe two different scenarios: a mobile device with intermittent connectivity to a central cloud, and a Mobile Cloud. For this architecture, the authors propose and describe Serendipity presented in [26]. For intermittent connectivity, the authors present an offloading algorithm for the case where the future network connectivity state is known. The application is modelled using the profile tree of the application as presented in [50]. The cost is modelled by weights in nodes and edges representing the execution and transmission costs respectively. In this solution, if an method is offloaded, then all the methods that are called directly or indirectly from said method are offloaded as well. In other words, if a node of the tree is offloaded, then all descendent nodes are offloaded as well. Therefore, in addition to the assumption of knowing the future network state, this proposal is limited to applications where all methods can be offloaded.

A Dynamic Offloading Algorithm for Mobile Computing Huang et al. [54] present an offloading algorithm for MCC. Authors model the application as a call graph with

N offloadable modules and one non-offloadable module similar to [10] where the weights correspond to time and energy costs of the modules. The optimization problem is formulated as a queueing network problem and solved as a Lyapunov optimization. The heuristic algorithm aims to minimize the average energy cost under a given execution time constraint. They performed simulations for a mobile user moving through an area divided in sub-areas, showing that their solution achieves better energy savings on scenarios with higher density of WLAN networks.

Reliability-based design optimization for cloud migration Qiu et al. [86] use call graphs to increase reliability of legacy application that are being migrated to the cloud. The application is represented as a call graph where each node represents a module or class and invocations are represented by the edges. The execution of the application is then a path over the graph. Each node has a probability of fault which is assigned randomly from a distribution taken from observations of real services. An execution has a fault probability which is computed using the fault probability of the nodes on the execution path, and on the existence of a fault tolerance strategy or not. To make the offloading decision, the modules of the application are ranked according to a probability of failure and the migration decision is chosen such that the reliability of the application is maximized. To evaluate their work, scale-free graphs that represent applications are generated, and executions are simulated by generating paths over the graphs.

Run Time Application Repartitioning in Dynamic Mobile Cloud Environments Yang et al. propose a VM-based offloading solution in [53]. Their method-based solution models the application using a method call tree. The method call tree is compared to the call graph model where a given method can only be invoked from on one other method. The solution does not allows nested migration, therefore if a method is migrated, all its descendant in the tree are migrated as well. To estimate the cost of offloading and running a method, the framework takes into account the device workload and the network upload and download bandwidths. The authors develop two algorithms, i.e., *offline* and *online algorithms*. In both algorithms, the optimization goal is to minimize the application execution time. The difference between both algorithms is that while the offline algorithm assumes that the future predictions of the device load and network bandwidth are given, the online algorithm considers the current progress of the application execution and the current mobile device load and network bandwidth. The proposal is evaluated using a face-detection and a QR-reader application. In their evaluations, the future network status is predicted from measurements obtained from 30 mobile trajectories on a university campus

with a set of access points and measuring the network bandwidth every 20 s. The authors showed that the framework provided up to 35 % better application performance compared to other approaches

Cloud-Assisted Computation Offloading to Support Mobile Services In [87], Elgazzar et al. propose an offloading solution where a user may offload computations to a cloud provider, and where data required by offloaded operations may be hosted in a third party provider. The application execution unit is an operation (method), where a set of operations form a *service*. The entities in the offloading problem definition are a *mobile user*, which issues operation offloading requests, the *cloud*, which executes offloaded operations, a *mobile service provider*, which is a mobile device acting as an integration point between the user and the cloud, and the *data provider*, which may host some data required by the mobile operations. Data on the data provider can be fetched directly by the cloud offloading provider through faster network links. The mobile service provider runs the offloading decision algorithm.

The decision/selection algorithm considers the resource status of the mobile devices, network conditions and the mobility of both user and mobile service provider, and selects from a set of predefined offloading plans. The decision algorithm chooses an offloading plan that provides the optimal performance among a set of plans that meet user constraints. The mobile provider may gather the response from the cloud before sending it to the user, or the cloud may send the response directly to the user.

The authors' emphasis is on the framework therefore some details are not clear regarding the application and cost models and how the set of candidate plans. Regarding the application model, authors mention that operations may depend on other operations, and that such information is used when profiling the total cost of every operation, similar to the CGM application model. It is not explained nor illustrated which type of dependencies are considered and how they are used in computing the total cost of an operation. Furthermore, each plan defines one execution decision per operation which may not be optimal for multiple executions of the same operation. Additionally, the profiled data accounts for average values such as CPU, memory and energy costs, therefore the cost model is average-based and it may be inaccurate for operations whose costs vary considerably between different executions.

Optimal Joint Scheduling and Cloud Offloading for Mobile Applications Mahmoodi et al. [88] use the call graph model together with a Integer Linear Programming (ILP) pro-

blem formulation to solve a join problem of scheduling and application offloading. Similar to MAUI [2], their objective is to maximize the energy saved by offloading while running the application under a deadline. The modules of the application are assigned to run either on the cloud or on mobile device. This work considers series and parallel dependencies between nodes which, together with diving the time in intervals, results in a problem formulation with a number of variables $O(Tn^2)$, where T is the number of time intervals and n is the number of modules. As a result, the offloading decision algorithm must run in the server. They performed simulations for a face detection application and compared the energy saved by their proposal to that of other methods such as no offloading, full offloading, [54] and [89].

Application Offloading based on R-OSGi in Mobile Cloud Computing In [90], Yang et al. propose an offloading scheme that leverages R-OSGi java framework. Similar to other works, the offloading problem is formulated as a combinatorial optimization problem aiming at minimizing the total execution time of the entire mobile application. The application is represented by a tree where nodes are java modules that may call each other. If a module in the application is called by more than one module, the node in the tree is cloned an the tree starting from said node is duplicated. There are important differences between this application model and the Execution Dependency Tree (EDT) model presented in [14]. First, the EDT model considers three types of relations between modules, i.e., parallel, series, and probabilistic, and, second, the EDT tree represents the actual execution paths observed during the execution of the application, instead of only static relations between modules.

Coalition-based Energy Efficient Offloading Strategy for Immersive Collaborative Applications in Femto-Cloud Yu et al. [42] consider the scenario where multiple mobile users offload duplicated computation tasks to a set of nodes. The goal is to minimize the energy consumption by offloading and sharing results, considering the delay constraint of each mobile device and the computation and memory constraints of each server. The problem is modeled by a cooperative weighted call graph. They present a distributed algorithm that combines notions from Binary Linear Programming (BLP) and coalitional game.

Fine-granularity Based Application Offloading Policy in Cloud-enhanced Small Cell Networks Deng et al. [91] address the scenario of multiple mobiles in a Cloud

Enhanced Small Cell Networks. The offloading policy aims at minimizing the energy consumption while satisfying a strict delay constraint. The application is modeled as a directed acyclic graph where nodes are tasks that can be offloaded. The problem is solved via a Binary Particle Swarm Optimizers (BPSO) algorithm to incur in low computing load.

Automated Application Offloading through Ant-inspired Decision-Making In [92], the application is modeled by a call graph whose nodes represent methods. The call graph is then transformed to account for the different offloading environments (mobile and cloud). The offloading problem is then equivalent to the shortest path (SP) problem on the new graph. An ant-inspired heuristic algorithm is used to solve the SP problem.

A fast hybrid multi-site computation offloading for mobile cloud computing In [93] propose a hybrid solution that includes two different decision algorithms to achieve the optimal and the near-optimal partitioning of small-scale and large-scale applications, respectively. The problem is formulated as a multi-site offloading problem. The application is modeled via a weighted call graph where each node has a weight per site and edges have a weight representing the transmission cost. The application cost is formulated as the sum of the cost of each unit or module of the application. The energy cost is computed from the time cost multiplied by constants that represent the energy consumption rate for the mobile's CPU and the network interface. The first algorithm uses the branch-and-bound algorithm with an optimal branching rule, a fast search strategy and an bounding function. The solution does not consider parallel applications, therefore the advantage of using multiple sites is reduced.

A summary of some of the approaches for application offloading problem is presented in Table 2.2.

Among the several approaches to the application offloading problem, none of them use robust optimization. However, in Chapter 5, we present a robust optimization formulation to the problem. Therefore, we briefly introduce the topic in the following section.

2.4 Robust Optimization

Application offloading is one of the optimization problems in science and engineering that are often uncertain due to, for example, inexact measurements of physical entities, parameter fluctuation and values that are only known in the future. Uncertainty may result in

Year	Article	Exec. Model	App. Model	Algorithm	Offloading Decision
2017	[81]	cs	s	H	d
2016	[88]	cs	c	LP	s
2016	[53]	vm	c	H	s/d
2016	[90]	cs	c	LP	s
2016	[42]	cs	c	H	s
2016	[91]	cs	c	H	s
2016	[92]	cs	c	H	s
2016	[93]	cs	c	H	d
2015	[82]	cs	s	H	d
2015	[83]	cs	s	O	s
2015	[52]	cs	s	O	s
2015	[82]	cs	s	H	s
2015	[45]	vm	-	-	-
2015	[84]	cs	s	O	d
2014	[79]	-	s	H	-
2014	[87]	-	c	H	semi-d
2014	[80]	cs	s	H	d
2013	[86]	cs	c	H	s
2012	[21]	cs	s	H	s
2012	[78]	cs	s	H	d
2012	[67]	vm	t	H	s
2012	[11]	vm	s	H	d
2012	[26]	cs	s	H	d
2012	[3]	cs	c	H	d
2011	[50]	vm	s	LP	s
2011	[68]	cs	c	H	d
2010	[94]	cs	c	-	d
2010	[12]	cs	s	H	d
2010	[24]	cs	s	-	d
2010	[2]	cs	c	LP	d
2008	[25]	cs	s	-	-
2006	[10]	cs	c	H	d
2003	[62]	cs	s	H	d
2002	[61]	cs	s	H	d

Table 2.2: Summary of some application offloading approaches. Execution Model: VM-migration (vm), Client-Server (cs). Application Model: SGM (s), CGM (c), Tree CGM (t). Algorithm: LP, Heuristic (H), Other (O). Offloading Decision: static (s), dynamic (d).

highly infeasible or suboptimal solutions as shown by Ben and Nemirovski [95]. They show that even 1% uncertainty for a set of problems affect the feasibility of the solution, and that the solution to these problems using the Robust Optimization methodology ([96–100]) looses nearly nothing in optimality.

2.4.1 Motivation

The robust methodology is often preferred over other approaches because of two main reasons: *set-based* uncertainty and *tractability*. First, the robust approach only requires the uncertainty to be specified as a set, which is the appropriate approach for certain optimization problems. This contrasts with the Stochastic Optimization approach which requires knowledge about the distribution of the possible values of the parameters.

Second, despite the problem is generally NP-hard due to an increased number of constraints resulting from the uncertainty, it has been shown that the robust problem is tractable under certain conditions, given by the structure of the space formed by the constraints and the uncertainty set [101]. Two examples are when the uncertainty forms a given uncertainty ellipsoidal set (Ben and Nemirovski [96]), and for the mixed ILP problem under uncertainty in the coefficients of the objective function (Atamtürk [102]). Bertsimas et al. [103] focused on robust optimization for discrete optimization and network flow problems, and propose an efficient algorithm that solves the robust problems in a tractable way, and whose degree of conservatism can be adjusted. The authors propose robust formulations and efficient algorithms for the problems when the uncertainty affects only the objective function, or only the constraints. In a similar approach, Bertsimas and Sim [104] address the level of conservatism proposing a new robust approach for lineal optimization problems that guarantees that the robust solution is feasible if less than a specified number of coefficients take worst case cost simultaneously. Otherwise, the solution is feasible with high probability. The robust formulations are lineal optimization problems and extend to discrete optimization problems in a tractable way.

Robust optimization has been successfully employed in a wide range of problems. Ben-Tal and Nemirovski [95] apply robust optimization to a set of Netlib LP problems and show that the robust formulations lose nearly nothing in optimality while being *immune* to parameters' uncertainty. In [97], Ben-Tal and Nemirovski solve the problem of designing a truss topology that maximizes the rigidity of the structure to forces in a given direction, while being robust to occasional loads in other directions. A robust inventory control problem is solved in [101, 105–108]. The goal is to design a system that makes ordering,

stocking and storage decisions in order to meet demand while minimizing costs. In [103], Bertsimas et Sim solve the robust counterparts of several discrete optimization problems. The authors solve a robust Knapsack problem where weights are uncertain, following a symmetrical distribution, and where the values of the objective function are deterministic. The formulation aims at maximizing the total value of the goods while allowing a maximum of 1% constraint violation. The authors then solve a robust formulation of Sorting when the elements are subject to uncertainty. The problem is formulated as an integer programming model with uncertainty in the objective function. They also solve the Shortest Path problem as a BLP problem where arcs are subject to uncertainty. Other problems include portfolio optimization [109] and binary classification [110].

2.4.2 Objective

The goal of the robust optimization problem is to find a solution that is feasible for any realization of the parameters in a given uncertainty set. Furthermore, the solution is considered optimal if it guarantees a minimum value for any realization of the parameters in the given set.

This formulation, however, has been criticized for being too conservative for some problems because it ignores near feasible or near optimal solutions. Consequently, other formulations have been proposed with different definitions of feasibility and optimality, adjusted to the nature of the original optimization problem and the characteristics of the parameters' uncertainty. Among these formulations, Bertsimas et al. [103] propose a robust formulation for combinatorial optimization problems that addresses both the uncertainty of the problem and the conservatism of the robust optimization.

Our work focuses on the Application Offloading problem which can be formulated as a combinatorial optimization problem. Therefore, we present here the robust formulation for these problems by Bertsimas et al. [103].

2.4.3 Robust Combinatorial Optimization

Consider the combinatorial optimization problem:

$$(p_{\boldsymbol{\rho}}) \qquad \begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{\rho}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in X \subseteq \{0,1\}^n. \end{aligned} \tag{2.1}$$

Consider that the coefficients $\boldsymbol{\rho} = [\rho_1, \ldots, \rho_n]$ of the optimization function take values in an interval, i.e., $\rho_i \in [\hat{\rho}_i, \hat{\rho}_i + d_i]$, $d_i \geq 0$, but the set X is fixed. For the application offloading problem, the parameters ρ_i account for the execution costs on the mobile, the cloud, and the transmission costs. These costs include uncertain environment conditions such as device CPU load and network data rate. This approach models other problems such as the shortest path, the minimum spanning tree and the traveling salesman. Data uncertainty in the shortest path problem for example, expresses the errors when measuring distances in some of the links of the graph.

There is no uncertainty in the constraints, therefore a solution $\boldsymbol{x}$ to the uncertain optimization problem (P) is considered *feasible* if $x \in X$.

The solution $\boldsymbol{x}$ to the uncertain optimization problem (P) is *optimal* given a value $0 \leq \Gamma \leq n$ if $\boldsymbol{x}$ is feasible and $\boldsymbol{x}$ guarantees the minimum value for all realizations of at most Γ parameters, and where the other $n - \Gamma$ parameters are certain. In other words, it should be an optimal solution to the following certain problem:

$$(P^*) \quad \min_{\boldsymbol{x}} \left(\hat{\boldsymbol{\rho}}^T \boldsymbol{x} + \max_{S, S \subseteq \{0,\ldots,m\}, |S| \leq \Gamma} \sum_{i \in S} d_i x_i \right) \quad \text{s.t.} \quad \boldsymbol{x} \in X \tag{2.2}$$

where $\hat{\boldsymbol{\rho}} = [\hat{\rho}_1, \ldots, \hat{\rho}_n]$. Consequently, the level of conservatism can be adjusting by specifying Γ. In particular, if $\Gamma = n$, the solution $\boldsymbol{x}$ is optimal if it guarantees the minimum cost for all realizations of the parameters in the uncertainty set. This is equivalent to the most conservative robust optimization. On the other hand, if $\Gamma = 0$, then the robust and the nominal problems are equivalent, and thus $\boldsymbol{x}$ is optimal if it is optimal for the nominal problem 2.1. This robust problem is tractable and can be solved by solving no more than $n + 1$ instances of the nominal problem [103].

To summarize, robust optimization may be used for optimization problems where parameters are uncertain. In Chapter 5, we will present a robust formulation of combinatorial problems with uncertainty in the objective function.

2.5 Summary

The goal of Mobile Cloud Computing is to enhance the experience of the mobile user, who benefits, for example, of faster application execution and improved device's battery life. The technology is implemented in three different architectures, namely, *Remote Cloud*

where the mobile user leverage the resources of a remote cloud, *Mobile Cloud* where mobile devices are organized and share their resources, and *Cloudlet* where a less powerful cloud is located near to the mobile device, reducing the latency between the server and the device. Implementing these architectures involves addressing many challenges such as the application offloading problem.

The application offloading problem aims at minimizing the cost of running the mobile application by offloading its execution, entirely or partially, to the cloud. While the offloading operation saves resources on the mobile by executing modules on the cloud, it also incurs in communication costs when transmitting the necessary information between the mobile device and the cloud. This tradeoff must be effectively evaluated in order to guarantee a better experience for the mobile user overall. The cost tradeoff depends on the execution model used for offloading, which can be performed via VM-migration, where the mobile environment is virtualized on the cloud server, or by Client-Server model, where offloading is performed via client-server protocols such as RMI and RPC.

The solution to the problem involves modeling the application and estimating the execution cost given an offloading decision. The two main application models have been the CGM and the SGM, in which nodes in a graph represent parts of the application and edge model calls between the modules. The main difference between both models is that, in the former, modules may execute other modules, whereas modules are independent and cannot call other modules in the later model. The cost model evaluates the tradeoff between the cost saved and the cost incurred by offloading operations. The cost of a module is modelled by the average of the costs observed in previous executions of the module, and is used to define the optimization function and constraints of the problem. The optimization function estimates the average execution cost of the entire application, and is used in a minimization problem by the offloading decision algorithm. The application and cost models are used by the offloading decision algorithm in order to find the offloading decision that minimizes the objective function while satisfying the constraints. Approaches differ on which techniques they follow, such as LP formulations, or when the decisions are made, i.e., static or dynamic algorithms, and where the algorithm is run, i.e., on the mobile device or the cloud server. We observed that most relevant solutions in literature are dynamic and run in the mobile, while both application models, SGM and CGM, are commonly used. The application offloading problem is highly affected by uncertainty from, for example, estimating the network data rate or CPU load.

Robust optimization is a methodology used to deal with set-based uncertainty in a tractable way. Versions of the Shortest Path and the Minimum Spanning Tree problems

with uncertainty are two of the problems solved via robust optimization. The method aims at finding a solution that is feasible and optimal in a robust sense, considering the uncertainty and the problem characteristics. In the typical approach, a solution is feasible if it is feasible for all the realizations of the parameters in the uncertainty set. Also, the solution is deemed optimal if it is feasible and guarantees the minimum value for all the realization of the parameters in the uncertainty set. For some scenarios, this definition is considered too conservative as it discards almost-feasible and almost-optimal solutions, thus some formulations also aim at controlling the degree of conservatism. While the uncertainty may result in virtually unlimited constraints, there are scenarios where the robust counterpart of an optimization problem can be solved efficiently. For example, problems in the family of combinatorial optimization, which include the application offloading problem, have robust formulations that are both tractable and whose degree of conservatism can be adjusted.

The existing approaches for application offloading suffer from inaccurate estimates and decisions caused by the limitations of the application and the cost models. On the one hand, the application models employed by literature do not support complex programming instructions, such as conditionals and parallel execution of modules, nor they capture the different execution paths that affect the cost of the application. On the other hand, the average-based cost model restricts the problem to optimizations of the average cost of the application and does not support statistical analysis of the costs of the application. Consequently, the offloading decision algorithm can only optimize average costs of the application, and makes a suboptimal offloading decisions. Another limitation results from uncertainty in the problem parameters. For instance, measuring the network data rate or the CPU load cannot be done without errors, furthermore, these conditions vary with time rendering offloading decisions infeasible or suboptimal, or both, and therefore worthless.

In the next chapters, we address these limitations proposing novel application and cost models. We also develop a robust solution for scenarios with limited, uncertain cost information.

Chapter 3

Novel Application Model

A key aspect in solving the application offloading problem lies on the application and cost models. A common limitation of state of art application models is that they do not consider the types of dependencies between the modules of the application nor they capture the execution paths of the module's executions. This leads to cost inaccurate cost models and thus offloading decisions.

In this chapter, we address these limitations proposing a novel application model that captures the execution paths of the application and the types of dependencies between the application's modules. The model supports multiple offloading decisions per module, as well as modeling the cost of the modules based on the execution paths. We present an average-based cost model and an efficient offloading decision algorithm based on the novel application model.

3.1 Application Model

We model a given mobile application using a tree structure that we refer to as the EDT. The model aims at overcoming the problem of imprecise estimation of the modules' execution cost. This problem is more critical for applications that contain modules exhibiting

significant variance in their execution cost as they are called within different execution paths of an application. This issue is elevated by incorporating two additional pieces of information that were not considered in previous approaches; the first is the module cost as a function of its execution path, relying on the use of module dependencies. The second is the probability distribution of the modules' cost.

More precisely, our proposed tree structure allows for the identification of three different execution relations, or dependencies, among two or more modules. The first is the *series dependency*, or sequential execution dependency that defines a specific order of the sequential executions among the modules. The second relation describes modules related through conditional branching statements. In this case only one module may be executed according to the satisfied condition. We refer to this relationship as a *probabilistic dependency*. Finally, the relationship among modules that can be executed in parallel, if sufficient resources are available, is referred to as a *parallel dependency*.

The application modules and their above dependencies are represented using a EDT structure $T = (V, E)$. The set of nodes $V = \{V_M \cup V_D\}$, $|V| = n$, is the union of two subsets; V_M is the set of the module nodes representing the application modules. We make no assumptions with respect to the granularity of these modules. They can represent functions, methods, threads or components chosen by the user. On the other hand, V_D is the set of dependency nodes that are used to describe one of three possible modules execution relations, namely, series, probabilistic and parallel relations.

A dependency node must be a parent of either module or dependency nodes. On the other hand, to maintain lucidity of our presentation, we allow a module node to only be a caller, i.e., a parent of a single dependency node. If a module node is the caller of two or more modules, then a dependency node is needed first to specify the relation between these modules. On the other hand, if the module node is self-contained, i.e., it does not call any other modules, it becomes a leaf node in the tree. Finally, the case of module with a call to a single module is represented by a call first to a series dependency node that in turn calls that module. This generalized approach will simplify the derivations of the rules needed to derive the overall cost of the tree. Finally, we use the term execution path of a module to refer to the sequence of nodes when the tree is traversed from the root to that module.

Figure 3.1 depicts the pseudo-code of a generic application and its corresponding EDT. The application is partitioned into modules a-f corresponding to its methods a-f. In the example, a conditional branching due to the if statement is translated to a probabilistic dependency. Similarly, it shows that d and e can be executed in parallel while the series

relation mandates that f must be executed before the conditional statement. We also note that as c is called twice and appears as c_1 and c_2, once within each execution path leading to that node. Hence, the EDT allows us to accurately profile the cost of the module c while taking into consideration the application's execution path. As will be shown later this can dramatically reduce the imprecision in estimating the module costs.

```
1  public void a()
2      f ();
3      if condition then
4          b ();
5          c ();
6      else
7          fork(d ());
8          fork(e ());
9      end
10 public void b()
11     c ();
```

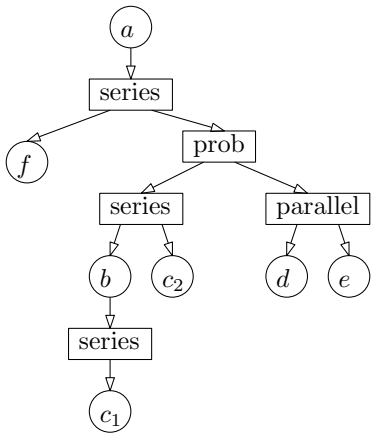


Figure 3.1: An example of an application and its Execution Dependency Tree.

This model is more detailed compared to the state of the art Call Graph Model. It supports different offloading decisions per module based on the execution path that results on the execution of the module. Additionally, it captures and uses implementation details of the application in order to further increase the precision of the estimation of the execution cost of the modules.

We restrict the number of modules that are considered for offloading, which must be specified by the developer. This is similar to other works such as MAUI [2] where the number of offloadable modules is limited, and Ou et al [10] where the reduction of the problem is performed by a $(k+1)$ partitioning algorithm. This avoids the Path Explosion problem as known in Symbolic Execution [111], [112], [113].

The EDT describes the structure of the application and its execution paths. Additionally, weights in nodes represent their execution and transmission costs and are used when finding the optimal offloading decision in a cost model as it is presented next.

3.2 Average-based Cost Model

The execution of a given module incurs two costs: a communication cost and an execution cost. The model is general enough so execution or transmission costs of a node may

express time, energy, monetary cost, etc., or a weighted combination of these. Each node in the tree is associated with two cost functions for costs resulting from communication and execution. The communication cost is determined by the location of both the module and the module's caller, and is negligible if a module runs in the same location as its caller. On the other hand, the execution cost depends only on the location of the module, i.e., at the mobile device or in the cloud environment.

Let $\boldsymbol{l} = (l_1, ..., l_n)$, $n = |V_M|$, be the location (offloading) decision vector where $l_i \in \{0, 1\}$, with $l_i = 0$ means that module v_i's location is the mobile device and $l_i = 1$ means the module's location is the cloud. We define two generic functions to express the communication and execution costs of nodes given a location decision.

Let $t_i(\boldsymbol{l})$ be the communication cost function for module v_i and location $\boldsymbol{l}$. This function takes into account the location of v_i and v_i's caller, the amount of data, and all environment variables like current network status and device properties.

Let $e_i(\boldsymbol{l})$ be the execution cost function that estimates the execution costs of module v_i when running in location $l_i \in \boldsymbol{l}$, and that considers all environment variables like device and server characteristics.

Based on these two functions, we define a function to express the total cost of running an application.

Definition 1. *Given the EDT representation $T(V, E)$ of an application and a location vector $\boldsymbol{l}$, $F(T_i, \boldsymbol{l})$ is defined as the Total Cost of running the application represented by T_i, where T_i is the subtree of T rooted at node $v_i \in V$.*

By definition, the cost associated with the root node v_1 represents the total cost of executing the application according to $\boldsymbol{l}$.

$F(T_i, \boldsymbol{l})$ is calculated as a combination of the transmission and execution costs of node v_i, combined with the total cost of its children trees, i.e., $F(T_j, \boldsymbol{l})$ where $v_j \in Children(v_i) \subset V$ are the child nodes of v_i in the tree T. Naturally, the computation depends on the type of v_i which can be either a module node or a dependency node. Expressions for each case are given next.

- If $v_i \in V_M$, then

$$F(T_i, \boldsymbol{l}) = t_i(\boldsymbol{l}) + e_i(\boldsymbol{l}) + \sum_{v_j \in Children(v_i)} F(T_j, \boldsymbol{l}) \tag{3.1}$$

i.e., the cost is the sum of the communication and execution costs of the node, plus the total cost of the applications it invokes.

- Series node, $v_i \in V_D$,

$$F(T_i, \boldsymbol{l}) = \sum_{v_j \in children(v_i)} F(T_j, \boldsymbol{l}) \tag{3.2}$$

i.e., the cost is the sum of the total cost of all the applications represented by the EDTs T_j for all v_j child of v_i.

- Probabilistic node, $v_i \in V_D$,

$$F(T_i, \boldsymbol{l}) = \sum_{v_j \in children(v_i)} P(v_j) * F(T_j, \boldsymbol{l}) \tag{3.3}$$

where $P(v_j)$ is the probability of running the application represented by T_j, $T_j \in Children(v_i)$. $P(v_j)$ is obtained by profiling the application and it is equal to the number of times execution follows to node v_j over the total times that node v_i is executed.

If the root node is a *parallel* node, then the accumulated cost depends on whether the cost unit is time or not. When measuring time, the total time taken by several parallel operations is the time of the longest operation.

- Parallel node, $v_i \in V_D$, when measuring time,

$$F(T_i, \boldsymbol{l}) = \max(F(T_j, \boldsymbol{l})) \tag{3.4}$$

where $v_j \in Children(v_i)$.

Otherwise, cost units such as monetary cost add to the final cost even if run in parallel, and thus the total cost is calculated as a series node.

- Parallel node, $v_i \in V_D$, otherwise,

$$F(T_i, \boldsymbol{l}) = \sum_{v_j \in Children(v_i)} F(T_j, \boldsymbol{l}) \tag{3.5}$$

which is the sum of the total cost of all children of node v_i.

The application is profiled on the server and on the mobile device. The resulting cost functions are stored by the server and provided to the application upon periodic pull requests or by push operations. Furthermore, similar to other approaches [2], the application can profile the network access to the cloud server periodically, or this information can by obtained during offloading requests, or be provided by some external API. As a result, the mobile application has up to date information about the application costs and network status, and runs the decision algorithm when the conditions change. As the number of offloadable modules is limited, finding the solution for the application offloading decision problem has negligible overhead.

3.3 Problem Formulation

Given the average-based cost model, we define the average-based application offloading problem as follows.

Definition 2. *The application offloading decision problem is defined as that of finding a location decision* $\boldsymbol{l}* = (l_1, ..., l_n)$ *such that* $F(T, \boldsymbol{l})$ *is minimized, where* T *is the EDT of the application.*

In other words, the problem is to find an offloading location for each application module such that the expected cost of the application is minimized. The expected application cost is computed according to (1) the modules and their dependencies as modelled by the EDT of the application, and (2) according to the average costs for each of them.

3.4 Proposed Offloading Decision Algorithm

To solve the problem for a given EDT T of an application and a node v_i, we note that the optimal location decision for $T_i \in T$ depends only on the location of v_i's parent. Therefore, the best location v_i is the best of two scenarios: (1) when v_i's parent is located in the mobile device and (2) when v_i's parent is located in the cloud. We use this fact to develop an optimal algorithm that solves the application offloading decision problem for an application represented by an EDT T.

The algorithm works in two steps as depicted in Algorithm 1. The first step *Solve*, shown in Algorithm 2, finds the best location for every node for each possible location of the node's parent. In other words, we find v_i's best location for each of both cases: (1)

v_i's parent is in the cloud and (2) v_i's parent is in the mobile device. Any restriction to the location of node v_i is considered here, for example, if v_i must only run in the mobile device.

The second step of the algorithm, *BuildSolution*, depicted in Algorithm 3, builds the optimum solution from the information from the previous step. The location of the root node, the starting module of the application, is by definition the mobile device. Given the location of the root node is the mobile device, the best location for the child nodes of the root node is chosen as computed in the first step. This process is then performed again for the child nodes of the just solved nodes, and it is repeated until reaching the leaves nodes of the EDT. At the end of the process, an offloading decision is made for each module, and the entire solution is the optimum offloading decision for the application for the specified EDT.

To analyze the cost of the algorithm, we note that in the first step, we visit each node in the tree once, from the leaves to the root. In the second step, the offloading decision is constructed by traversing the tree from root to the leaves, choosing at each step the final decision of each node. As a result, each node is visited twice, which yields an order $O(n)$ algorithm, where n is the number of nodes in the tree. The memory cost of the algorithm is $O(n)$ as only four values are kept per node: two location decisions (one for each possible value of the node's parent), and two total accumulated cost of each decision.

The proposed model is more accurate with respect to module costs, it supports multiple offloading decisions per module, and also provides a fast algorithm that can be run in the mobile device when required with negligible overhead.

Algorithm 1: Solution algorithm

```
Data: Dependency-execution tree T
Result: Location l* = [l_o, l_1, ..., l_n]
1 begin
2     Solve(T);
3     BuildSolution(T, 0);
4     return l*;
5 end
```

3.5 Performance Evaluation

We evaluate our offloading framework as a method-based granularity solution over the Microsoft .Net 4.5 Framework following MAUI [2]. We test our solution using a face-detection mobile application equivalent to the example of Figure 3.2. Other works that

Algorithm 2: Solve

```
   Data: Dependency-execution tree T
1  begin
2      r ← T.root ;
3      if solved[r] then return;
4      solved[r] = true;
5      foreach childTree ∈ r.children do
6          Solve (childTree);
7      end
8      decision[r][j] = i : (min(F(i, j) ∧ i, j ∈ {0, 1});
9      cost[r][j] = F(i, j) : j ∈ {0, 1});
       /* where F(i, j) is the cost function as expressed in Equations from 3.1 to 3.5
          corresponding to type of r, i.e., leaf, series dependency, etc.          */
10 end
```

Algorithm 3: BuildSolution

```
   Data: T: Dependency-execution tree
   Data: l_c: Caller's location
1  begin
2      r ← T.root ;
3      nodeLoc ← data[r][l_c];
4      l ∗ [r] ← nodeLoc;
5      foreach childTree ∈ r.children do
6          BuildSolution (childTree, nodeLoc);
7      end
8  end
```

also use image processing applications are Clonecloud [50] and Scavenger [12]. The former is evaluated using an image search application and the latter with a sequence of three image processing operations that is performed 50 times. Our experiment consists of performing full image face-detection and thumbnail face detection for 10 random images.

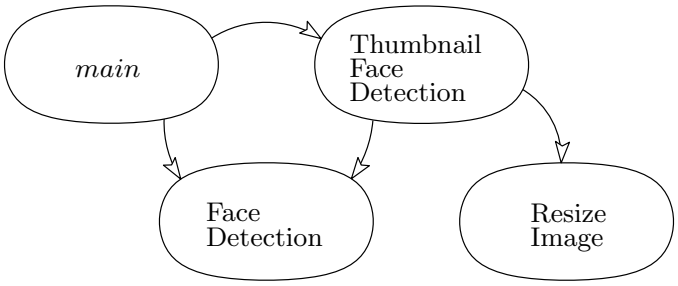


Figure 3.2: Call Graph Model of the FD application.

We compare the time taken by the *proposed* solution against three mechanisms: *traditional* schemes with a single cost/decision per module, *full offloading*, and *no offloading* decisions. The experiments were run under different network data rate conditions.

We simulate an HTTP RESTful offloading server that receives and executes module execution requests. Each request has all the information required to perform the offloading operation. Such information comprises the object, the module id and the method arguments. The module id includes the execution path leading to the current call. The server can simulate different network scenarios adjusting the data rate at which POST requests and responses are read and written by the server. The server also has a copy of the mobile application in order to execute the request. Information is sent between the mobile device and the server using binary serialization as it provides lower transmission costs than other formats, like Json for example.

Figure 3.3 shows the EDT of the application. The mobile application has three offloadable modules: *Face Detection* (*fd*), *Thumbnail Face Detection* (th), and *Resize Image* (rz). *Face Detection* can be called from two different execution paths. First, when *fd* is invoked from module main, face detection is performed directly on a full size image, and is referred as fd0. The second execution path is when *fd* is run from module *Thumbnail Face Detection*, which first resizes the target image and later performs face detection on the resized image; *fd* is then referred as fd1.

We profile the application using a sample of 200 images taken from the same smartphone, measuring execution cost of running the application on both the mobile device and on the cloud, and also measuring the transmission cost of offloading each offloadable module.

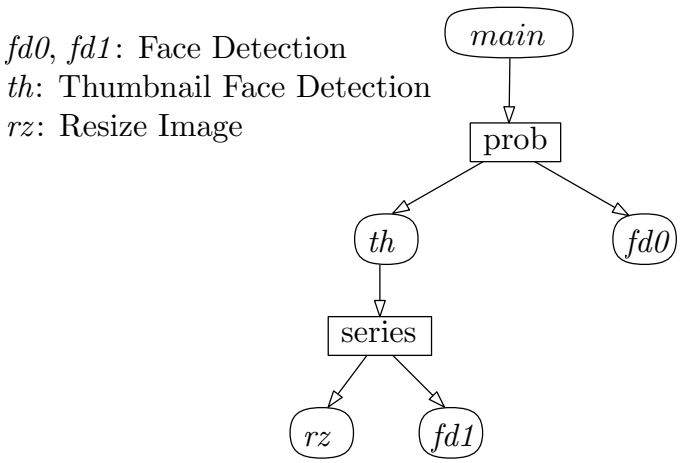


Figure 3.3: Execution Dependency Tree of the FD application.

Figures 3.4 and 3.5 show boxplots of communication and transmission costs, respectively.

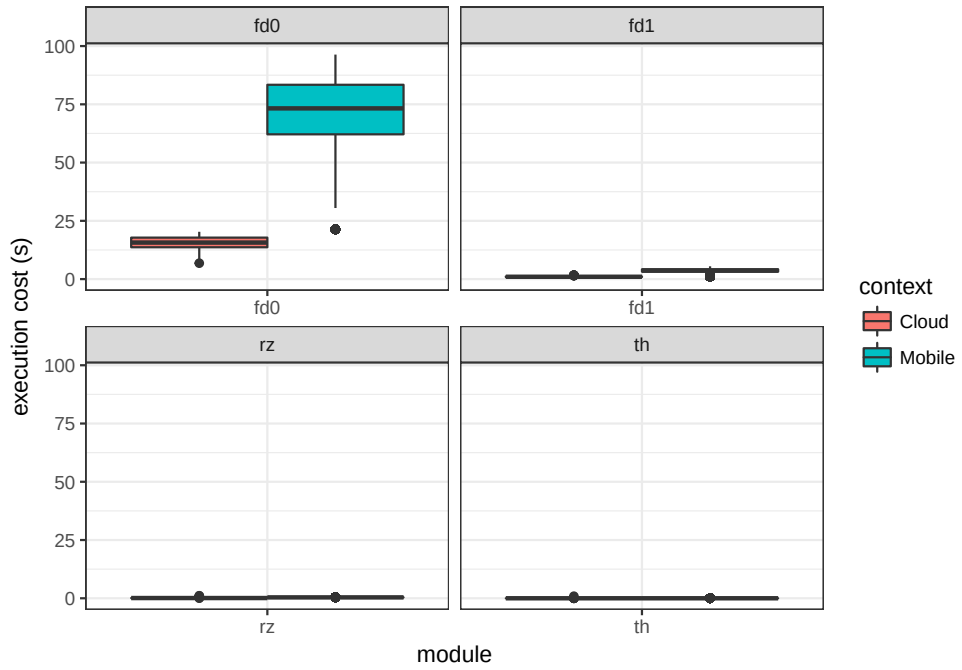


Figure 3.4: Execution cost of the modules of the FD application. The costs of nodes fd0 and fd1 differ even though they refer to the same module fd. The reason is significant cost difference for module fd due to different execution paths.

We can see that the median execution cost of fd0 and fd1 in the mobile device is 73.26 s and 3.65 s, respectively. Similarly, the transmission cost of said modules is 4.2 MiB and 0.26 MiB, respectively, for same execution paths.

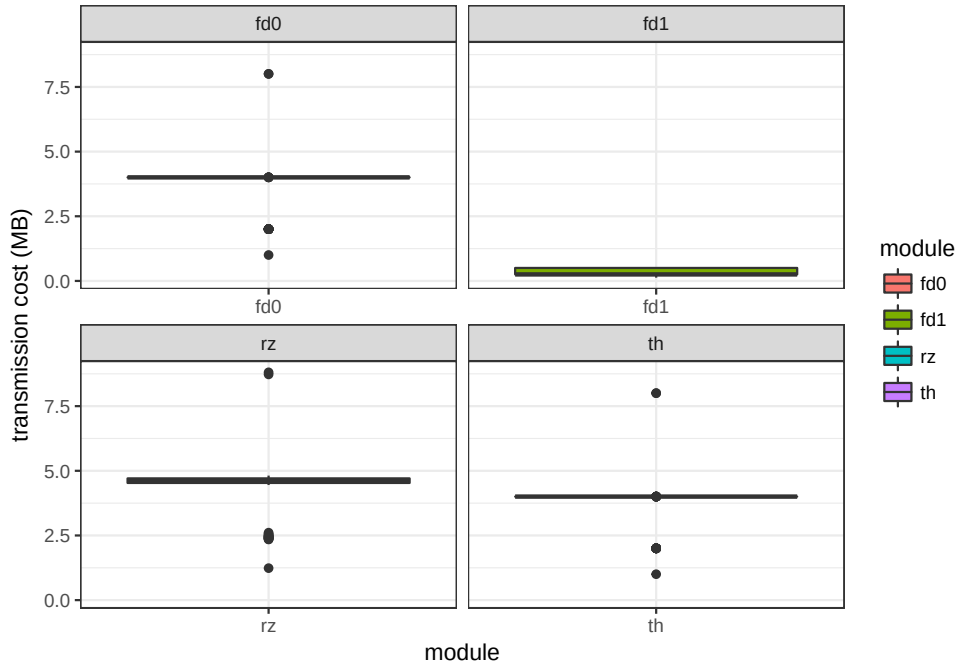


Figure 3.5: Transmission cost of the modules of the FD application. Nodes fd0 and fd1 shows significant cost difference, even though they both refer to module fd, due to different the execution paths.

Additionally, the median execution cost for fd0 in the mobile device is 73.26 s compared to 15.66 s the cloud. The main reason is that face detection module makes full use of the 8 parallel threads capability of the server. The costs of a module are estimated by the median value of the set of profiled costs as in our experiments it showed to be more stable than the mean value.

We also note important differences in the transmission cost for the module *Face Detection*. We see that the transmission costs of fd0 is around 4.2 MiB, while fd1's median cost is 0.26 MiB. This corresponds to a larger image of full resolution of 2592×1936 pixels for fd0 compared to a 640×480 pixels resolution used in fd1 after resizing.

If we follow the approach of using a *traditional* path independent cost per module, then the estimated execution and transmission costs of the modules are shown in Figures 3.6 and 3.7 respectively. This corresponds to the median value of all the profiled costs for *fd*.

Figure 3.8 shows the average execution cost in seconds for the application simulation over a range of network speeds. The figure shows that for network data rates of 0.07 MiB/s or lower, both approaches have a similar performance resulting from the decision of running everything in the mobile device. At 0.085 MiB/s, the proposed solution finds that fd0 must

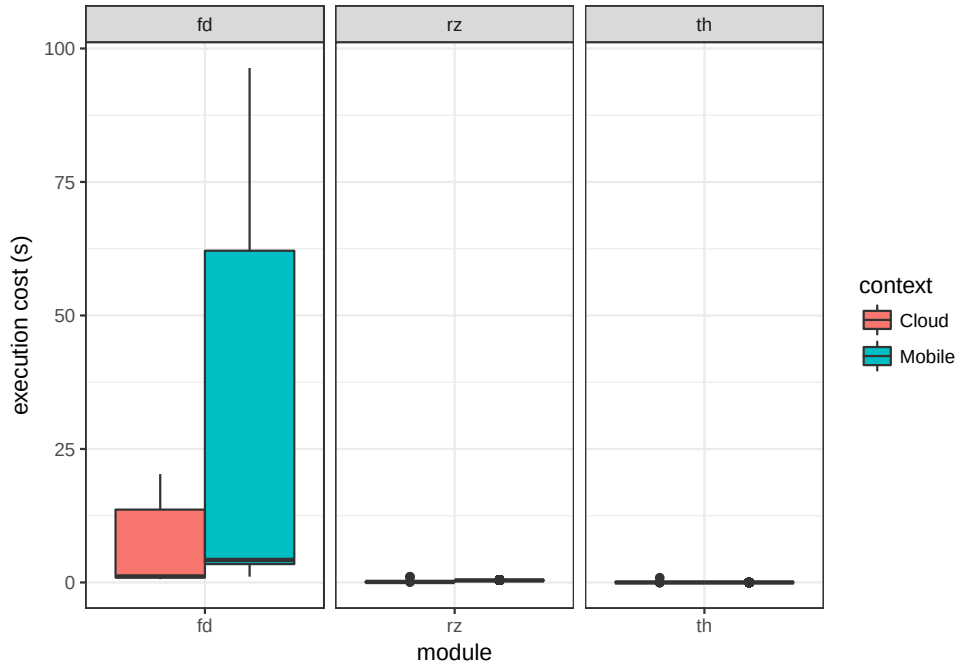


Figure 3.6: The profiled execution cost of the modules for traditional execution path independent approaches.

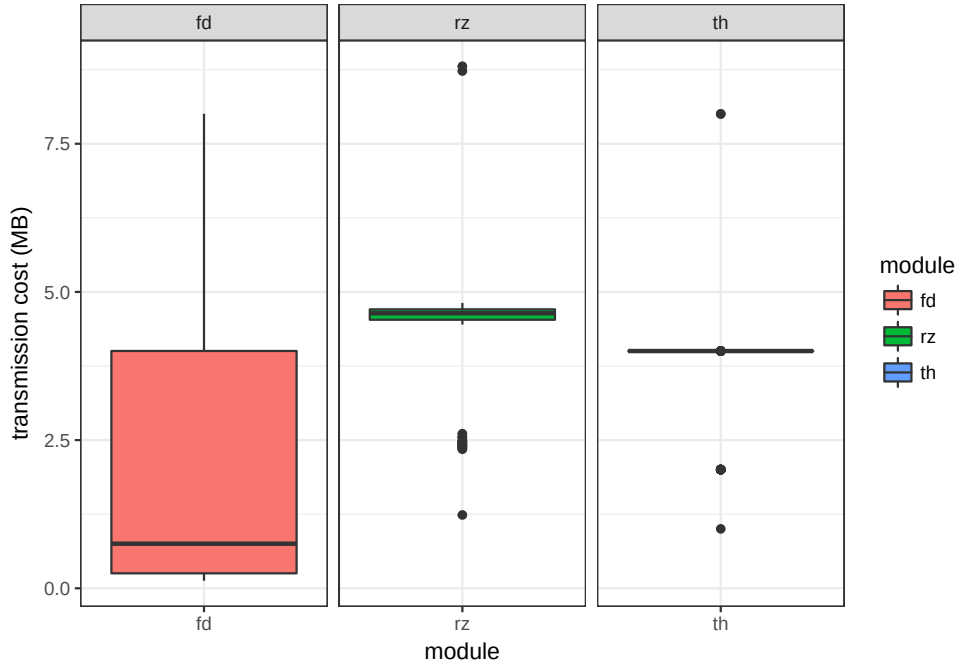


Figure 3.7: The profiled transmission cost of the modules for traditional execution path independent approaches.

be offloaded, and at 0.2 MiB/s, it decides that also fd1 must be offloaded. As a result, the proposed approach shows up to 42% faster application execution at network speed of 0.3 MiB/s. It is not until the data rate is at 0.5 MiB/s or faster that the *traditional* offloading solution finds suitable to offload *fd*, matching the solution from the *proposed* approach. For data rates of more than 1 MiB/s and up to 2 MiB/s, both approaches showed similar performance for the example analyzed. Using multiple costs estimation per module based on the execution paths improves application performance.

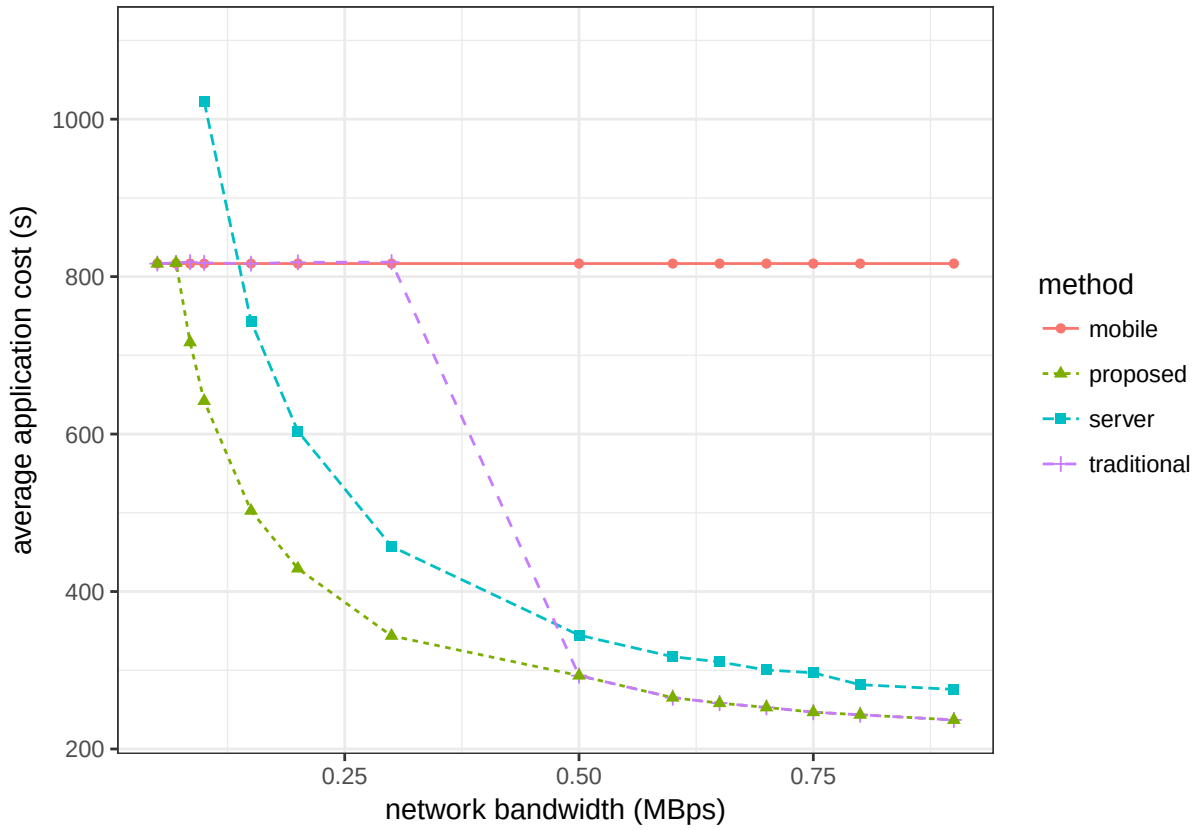


Figure 3.8: Offloading decision and average execution time for different network conditions.

3.6 Summary

This chapter described a novel application model and a new algorithm to solve the application offloading decision making problem. The novel model employs tree structures, referred to as *Execution Dependency Tree*, to accurately represent the different execution paths of the application. In contrast to existing models, this adopted structure allows an offloading algorithm to create multiple offloading decisions per module, or application component,

based on its current execution path. We evaluated our offloading framework and algorithm and compared its performance against the traditional single cost and decision per module for a face-detection mobile application under different network conditions. The new model achieved better performance for an interval of network data rate with up to 42 % time reduction for the tests.

In the next chapter, we will address the limitation of existing models that assume fixed average offloading costs. We propose a novel statistics-based cost model and generalize the offloading cost optimization functions to use statistical measures such as cost percentiles.

Chapter 4

Application Offloading via Statistical Analysis

The cost model plays a key role when evaluating candidate offloading decisions. The traditional average-based cost model described in the previous chapter produces estimates of the average execution cost. While this is appropriate for estimating average costs, this is still a limitation as the average value does not express the execution costs that may result from different executions of the application.

In this chapter, we address this limitation by proposing in Section 4.1 a statistics-based cost model that is not only more precise, but also more flexible as it supports optimization functions based on statistical measurement other than the mean. In Section 4.2, we formulate the offloading problem under the new cost model. It is followed by the offloading decision algorithm presented in Section 4.3. The new cost model allows new statistical analysis of the application costs. We present a method for simulating application executions costs via Bootstrap simulations using the novel cost model in Section 4.4. Finally, in Section 4.5, we evaluate the accuracy of the proposed model followed by a presentation of use cases for new statistical cost analysis of mobile applications.

4.1 Statistical Cost Model

The offloading problem for a given applications' EDT $T = \{V, E\}$, is to decide for each module whether it should be executed locally or offloaded to the cloud. Formally, the offloading problem is to find a location vector $\boldsymbol{l} = (l_1, l_2, \ldots, l_n)$, where each $l_i \in \{M, C\}$ represents the decided location of module $v_i \in V$. Here, setting $l_i = M$ or $l_i = C$ means that the module v_i will be executed in the mobile device or offloaded to the cloud, respectively. In other words, the location vector defines the final outcome of the offloading algorithm which determines the execution location, mobile or cloud, of every node in the EDT. This vector must be selected such that it satisfies a given offloading cost minimization objective as will be discussed below.

We first associate with each module node in the set V_M with one or more random variables that express the cost of executing the module. The choice of these random variables is tied to the overall objective of the offloading procedure. These variables can, for example, measure the amount of time it takes to execute the module, the needed CPU cycles, network communication overhead, or the consumed energy during its execution if the user is interested in minimizing the application response time, save the CPU cycles, the communication cost or reduce the consumed energy by the application, respectively. More than one variable can be used if the user desired to adopt a multi- objective function [114], [115].

Formally, we associate with each module node, v_i, two random variables X_i^M and X_i^C, that describe the execution cost, related to a given objective, when the module is executed locally on the device or offloaded to the cloud, respectively. Since transmitting data also consumes the mobile device resources (e.g., transmission time, energy needed for transmission and processing), we also associate with v_i, another variable X_i^T to model the communication cost needed to perform offloading operations. This cost is incurred when, for example, sending the input and output data of the offloaded module to and from the cloud.

Clearly, the cost of executing v_i must depend on its execution location as well as the location of its calling parent. Let $X_i|\boldsymbol{l}$ be another random variable that measures the module execution cost of v_i given a location vector $\boldsymbol{l}$, then

$$X_i|\boldsymbol{l} = \begin{cases} X_i^M & l_i = l_j = M \\ X_i^C & l_i = l_j = C \\ X_i^T + X_i^C & l_i = C, l_j = M \\ X_i^T + X_i^M & l_i = M, l_j = C \end{cases} \tag{4.1}$$

where *module node* $v_i \in V_M$ has a parent node $v_j \in V$ and l_i and l_j are their respective locations. The first two cases, in the above equation, require no communication cost between the parent and child modules as they are executed at the same location. Hence, the execution cost of v_i is that of either at the mobile device or the cloud. On the other hand, the last two cases, i.e., when $l_j \neq l_i$, the cost is a combination of both the communication cost between the parent and the child nodes in addition to the execution cost the module v_i.

Having calculated the execution cost of a single module, X_i, we need next to consider the subtree cost, Y_i, of executing all the modules in the subtree with v_i as its root. We refer to this cost as the *subtree cost* of v_i. This cost includes its own execution cost X_i as well as that of all its children. Let $Ch(v_i) \subset V$ be the set of children module nodes of node v_i, we can calculate the subtree costs as follows.

If v_i is a leaf node in the tree, then Y_i is simply the module cost of the node, that is,

$$Y_i|\boldsymbol{l} = X_i|\boldsymbol{l}, v_i \in V_M, \text{s.t. } Ch(v_i) = \emptyset. \tag{4.2}$$

On the other hand, if v_i is a module node with a single child v_k, the subtree cost $Y_k|\boldsymbol{l}$ of the child node v_k must also be included in the subtree cost of the parent node v_i. In this case, Y_i is computed as follows.

$$Y_i|\boldsymbol{l} = X_i|\boldsymbol{l} + Y_k|\boldsymbol{l}, v_i \in V_M, \text{s.t. } Ch(v_i) = \{v_k\} \tag{4.3}$$

The subtree cost for dependency nodes is computed according to the dependency type as follows.

If v_i is a series node, then its subtree cost can be calculated as the sum of the costs of its subtrees. That is:

$$Y_i|\boldsymbol{l} = \sum_{v_k \in Ch(v_i)} Y_k|\boldsymbol{l}, v_i \in V_D. \tag{4.4}$$

If v_i is a *probabilistic* node, define $P(v_k)$ to be the probability of executing $v_k \in Ch(v_i)$, $P(v_k) \in [0, 1]$, such that $\sum_{v_k \in Ch(v_i)} P(v_k) = 1$. Then, Y_i is a linear combination of random variables Y_k with weights $P(v_k)$, i.e.:

$$Y_i|\boldsymbol{l} = \sum_{v_k \in Ch(v_i)} P(v_k) \times Y_k|\boldsymbol{l}, v_i \in V_D \tag{4.5}$$

If v_i is a *parallel* node, then the subtree cost will depend on the type of the measured cost. For example, if the measure cost is the execution time, then it will be equal to the *maximum* or *largest order statistic* of the random variables of the subtree costs of its children nodes. In other words, in this case, if the modules are executed in parallel, then the completion time will be equal to the longest time taken by one of the module child subtrees.

$$Y_i|\boldsymbol{l} = \max_{v_k \in Ch(v_i)} Y_k|\boldsymbol{l}, v_i \in V_D \tag{4.6}$$

On the other hand, if energy is the measured cost, then the cost formulation will be similar to that of (4.4).

Now, define the function $F_X(x) \in [0, 1]$ to be the CDF of a given random variable X. Here, $F_X(x)$ the probability that the random variable X takes a value less than or equal to $x \geq 0$, i.e., $F_X(x) = P(X \leq x)$. Using this notation, define $F_{X_i^M}(x)$, $F_{X_i^C}(x)$ and $F_{X_i^T}(x)$, to be the CDFs for the variables X_i^M, X_i^C and X_i^T, respectively. Also define $F_{X_i}(x|\boldsymbol{l})$ and $F_{Y_i}(x|\boldsymbol{l})$ to be the CDFs of X_i and Y_i, respectively, given a location vector $\boldsymbol{l}$.

It is worth noting this work does not impose any restrictions on the specific functional forms of the CDFs.

Clearly, the above equations can be generalized by replacing the random variables with

their CDFs [55]. It can easily be shown that,

$$F_{X_i}(x|\boldsymbol{l}) = \begin{cases} F_{X_i^M}(x) & l_i = l_j = M \\ F_{X_i^C}(x) & l_i = l_j = C \\ F_{X_i^T} \otimes F_{X_i^C}(x) & l_i = C, l_j = M \\ F_{X_i^T} \otimes F_{X_i^M}(x) & l_i = M, l_j = C \end{cases} \tag{4.7}$$

where $\otimes$ is the convolution operator [116] and is defined between two functions F_j, F_k over a finite range $[0, x]$ such that

$$F_j \otimes F_k(x) = \int_0^x F_k(x-y) dF_j(y). \tag{4.8}$$

For the last two cases in Eq. (4.7), the cost X_i is given by the sum of the two independent random variables (Eq. (4.1)). Hence, the resulting CDF is given by the convolution their CDFs [116].

The CDF of the subtree cost of any node v_i is then computed as follow. If v_i is a leaf module node, then

$$F_{Y_i}(x|\boldsymbol{l}) = F_{X_i}(x|\boldsymbol{l}) \tag{4.9}$$

If v_i is a *module node with a child* node v_k, its cost is given by Eq. (4.3), and its CDF is computed as follows:

$$F_{Y_i}(x|\boldsymbol{l}) = F_{X_i}(x|\boldsymbol{l}) \otimes F_{Y_k}(x|\boldsymbol{l}). \tag{4.10}$$

If v_i is a *series* node, Y_i is defined by Eq. (4.4) and its CDF is computed by the convolution of the CDF of the $v_k \in Ch(v_i)$, i.e.:

$$F_{Y_i}(x|\boldsymbol{l}) = \otimes_{v_k \in Ch(v_i)} F_{Y_k}(x|\boldsymbol{l}). \tag{4.11}$$

Figure 4.1 depicts an example of the CDFs of random variables corresponding to the time cost of three modules, v_1, v_2 and v_3, with their costs given by Y_1, Y_2, and Y_3, respectively. The first two modules could represent some image processing operations (e.g.,

resizing and gray scaling). The random variable Y_3 represents the cost of applying both operations v_1 and v_2 in sequence, i.e., $Y_3 = Y_1 + Y_2$. In the example, the probability that each of v_1 and v_2 takes less than 50s is very low, consequently, the probability that v_3 is smaller than 100s is very low too.

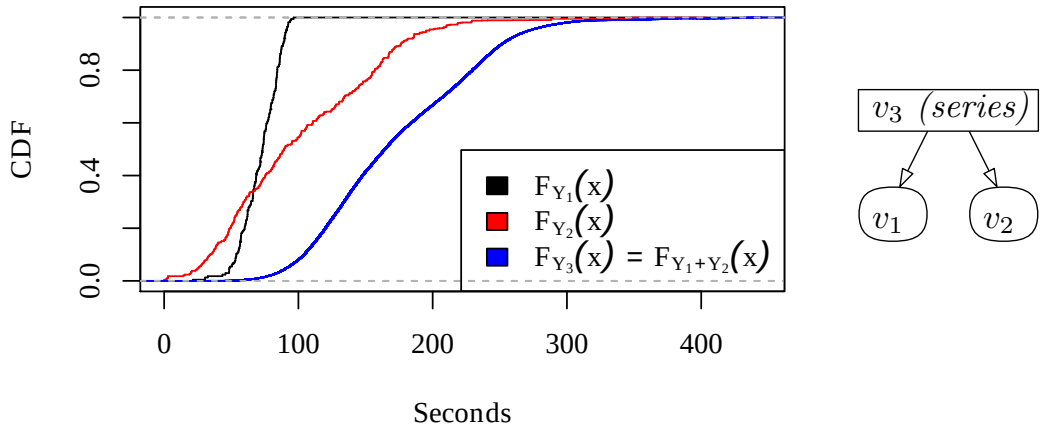


Figure 4.1: Cost CDFs of a series node v_3 and its children v_1 and v_2.

If v_i is a *probabilistic* node, the CDF of v_i is given by the CDF of X_i from Eq. (4.5), i.e.:

$$F_{Y_i}(x|\boldsymbol{l}) = \sum_{v_k \in Ch(v_i)} P(v_k) \times F_{Y_k}(x|\boldsymbol{l}). \tag{4.12}$$

In the example, suppose that operation v_3 is defined as running v_1 with probability 0.5, and v_2 otherwise, i.e., $Y_3 = 0.5 \times Y_1 + 0.5 \times Y_2$, Figure 4.2. It can be seen that the probability of v_3 taking a value equal to or smaller than x is equal to the average of the probabilities of each v_1 and v_2 taking x seconds or less, i.e., $F_{Y_3} = 0.5 \times F_{Y_3} + 0.5 \times F_{Y_2}$.

If v_i is a *parallel* node, then the CDF of X_i from Eq. (4.6) is calculated as:

$$F_{Y_i}(x|\boldsymbol{l}) = \prod_{v_k \in Ch(v_i)} F_{Y_k}(x|\boldsymbol{l}), \tag{4.13}$$

In the example, suppose we have operation v_3 that executes operations v_1 and v_2 both on the original image, and creates a new image with both results next to each other.

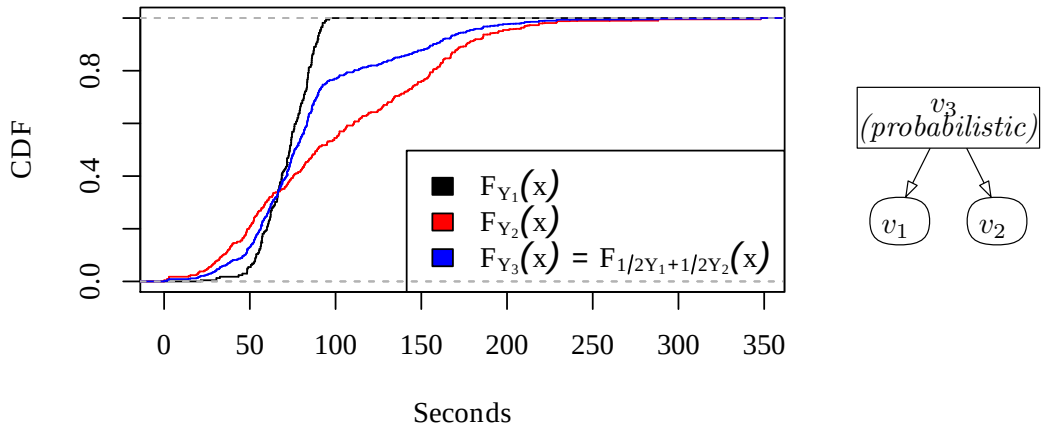


Figure 4.2: Cost CDFs of a probabilistic node v_3 and its children v_1 and v_2 with equal probabilities.

Operations v_1 and v_2 are performed in parallel and thus $Y_3 = max(Y_1, Y_2)$, Figure 4.3. In the example, the probability of v_3 finishing before 50s is almost zero as both v_1 and v_2 are unlikely to finish before 50s. Then, for about 80s, v_1 will almost certainly be completed, therefore, the probability of v_3 finishing before $x > 80$s is equal that of v_2.

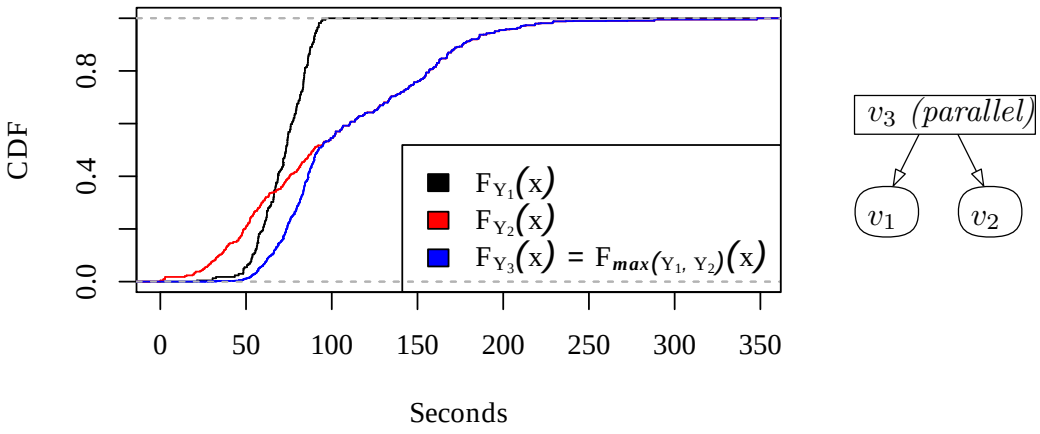


Figure 4.3: Cost CDFs of a parallel node v_3 and its children v_1 and v_2.

To derive a generic formulation for any subtree cost, we note that Eqs. (4.9)-(4.13) all

indicate that the subtree cost for a node is the module cost of its root module in addition to a function of the costs of its subtrees. We can hence derive a general formulation for these equations as follows,

$$F_{Y_i}(x|\boldsymbol{l}) = F_{X_i}(x|\boldsymbol{l}) \otimes \zeta_i(x|\boldsymbol{l}), \tag{4.14}$$

where $\zeta_i(x|\boldsymbol{l}) =$

$$\begin{cases} 0 & v_i \in V_M \text{ and } Ch(v_i) = \emptyset \\ F_{Y_k}(x|\boldsymbol{l}) & v_i \in V_M \text{ and } Ch(v_i) = \{v_k\} \\ \otimes_{v_k \in Ch(v_i)} F_{Y_k}(x|\boldsymbol{l}) & v_i \in V_D \text{ series} \\ \sum_{v_k \in Ch(v_i)} P(v_k) \times F_{Y_k}(x|\boldsymbol{l}) & v_i \in V_D \text{ probabilistic} \\ \prod_{v_k \in Ch(v_i)} F_{Y_k}(x|\boldsymbol{l}) & v_i \in V_D \text{ parallel} \end{cases} \tag{4.15}$$

Finally, the following definition demonstrates how the CDF of an application execution cost can be calculated using its EDT.

Definition 3. *Given the EDT $T = (V, E)$ of an application and a location vector $\boldsymbol{l}$, then $F_T(x|\boldsymbol{l})$, the cost distribution function for T given $\boldsymbol{l}$, is equal to that of the root node of T.*

The calculated CDF for an application can be interpreted according to the measured cost. For example, when the modelled cost is the execution time T, then $F_T(x|\boldsymbol{l})$ is the probability that T finishes by time x, and the mean value of the CDF is the expected time of completion of the application if its modules are offloaded according to the location vector $\boldsymbol{l}$. Similarly, when energy is the considered measure, then the CDF is the probability of the application consuming less or equal than x energy units.

4.2 Problem Formulation

As stated before, the offloading problem is to find a location decision $\boldsymbol{l}$ vector that minimizes the cost of running the application. In this section, we reformulate the problem in terms of the previously obtained cost CDFs for different location vectors. The comparison between these CDFs is mathematically defined by a binary relation over their set. More formally, let $\leq_D$ be a *binary relation* defined over the set of an EDT CDFs, $\mathcal{D}$.

The relation $\leq_D$ can be defined using any statistical measurement of the CDFs. For example, the traditional offloading problem definition aims at minimizing the average cost of the application. This is equivalent to defining $\leq_D$ as $F_i \leq_D F_j \iff E[X_i] \leq E[X_j]$, where $E[X]$ is the expected value of X. Similarly, $\leq_D$ can represent a function, for example, of the percentiles of the CDFs or the confidence intervals of the median.

Let $\mathcal{D}_\mathcal{T} \subseteq \mathcal{D}$ be the set of feasible CDFs for EDT T, i.e., the set of CDFs that can be obtained for T for all possible location decision vectors. The offloading problem is defined as follows.

Definition 4. *Given an EDT T of an application and $\leq_D$ a total binary relation defined on $\mathcal{D}$, the offloading problem is to find a location vector $\boldsymbol{l}*$ such that $F_T(x|\boldsymbol{l}*) \in \mathcal{D}_\mathcal{T}$ is infimum for the ordered set $(\mathcal{D}_\mathcal{T}, \leq_D)$.*

Since modules in the EDT can, in general, be executed locally or offloaded, the number of solutions for the offloading problem increases exponentially as the number of the EDT nodes increases. However, in some scenarios, the size of the tree is relatively small as the number of nodes to be considered is small and defined by the developers [2], [11]. In these situations, an exhaustive search can be used to obtain the optimal offloading location vector.

4.3 Proposed Offloading Decision Algorithm

In this section, we focus on applications where their cost CDFs for different location vectors have a total order under $\leq_D$. We show that, for these applications, the problem can be formulated as a set of subproblems with a similar structure and, hence, be solved efficiently using a simple dynamic program.

The key idea of the use of dynamic programming as a tool for optimization is to formulate the given problem using a form known as the Bellman equation [117]. The optimal solution of this equation is described using that of a set of subproblems. The optimal solutions of the subproblems must always lead to that of the original problem. Problems satisfying this property are said to have an *optimal substructure* [118]. Furthermore, in this formulation, solving the subproblems requires only solving one or more common subproblems. This property is referred to as having *overlapping subproblems* [118]. In turn, the subproblems are solved only once and their solution is reused repetitively.

Indeed, our problem formulation, Eq. (4.14), is a Bellman equation if it satisfies the optimal substructure property discussed above. This property requires that if the cost

$F_{Y_i}(x|\boldsymbol{l})$ is optimal, then the cost of the subproblems $F_{Y_k}(x|\boldsymbol{l})$ must be optimal as well for all $v_k \in Ch(v_i)$. The property must hold for each possible alternative of the term $\zeta_i(x|\boldsymbol{l})$ in Eq. (4.15), i.e., for each node type in the equation.

If v_i is a *series* node, the optimal substructure property holds if and only if choosing a location vector that leads to a better or equal cost for a child node v_k must also result in a better or equal cost distribution for v_i. We verify this property in the case of a node with two children for clarity of presentation, but the results hold for any number of children.

Formally, given a series node $v_i \in V_D$ with two children $v_k, v_m \in Ch(v_i)$ and let $\boldsymbol{l'}$ and $\boldsymbol{l''}$ be two alternative location vectors for the EDT rooted at node v_k. Also, fix $\boldsymbol{l}$ as the already selected location vector for the tree rooted at v_m, then the optimal substructure property holds whenever we have:

$$\begin{aligned} F_{Y_k}(x|\boldsymbol{l'}) \leq_D F_{Y_k}(x|\boldsymbol{l''}) \Leftrightarrow \\ [F_{Y_k}(x|\boldsymbol{l'}) \otimes F_{Y_m}(x|\boldsymbol{l})] \leq_D [F_{Y_k}(x|\boldsymbol{l''}) \otimes F_{Y_m}(x|\boldsymbol{l})] \end{aligned} \tag{4.16}$$

A similar reasoning can be followed for probabilistic nodes. Formally, given $v_i \in V_D$ a probabilistic node with two children $v_k, v_m \in Ch(v_i)$ with probabilities P_k and P_m, respectively, then the optimal substructure property holds whenever the following property holds:

$$\begin{aligned} & F_{Y_k}(x|\boldsymbol{l'}) \leq_D F_{Y_k}(x|\boldsymbol{l''}) \Leftrightarrow \\ & [P_k \times F_{Y_k}(x|\boldsymbol{l'}) + P_m \times F_{Y_m}(x|\boldsymbol{l})] \\ & \leq_D [P_k \times F_{Y_k}(x|\boldsymbol{l''}) + P_m \times F_{Y_m}(x|\boldsymbol{l})] \end{aligned} \tag{4.17}$$

For parallel nodes with the cost represented in Eq. (4.13), the property holds when:

$$\begin{aligned} F_{Y_k}(x|\boldsymbol{l'}) \leq_D F_{Y_k}(x|\boldsymbol{l''}) \Leftrightarrow \\ [F_{Y_k}(x|\boldsymbol{l'}) \times F_{Y_m}(x|\boldsymbol{l})] \leq_D [F_{Y_k}(x|\boldsymbol{l''}) * F_{Y_m}(x|\boldsymbol{l})] \end{aligned} \tag{4.18}$$

we can now enunciate the following lemma.

Lemma 1. *Let T be the EDT of an application, and let $(\mathcal{D}, \leq_D)$ be a total ordered set representing the total ordered set of its cost CDFs, then the offloading problem can be efficiently solved by dynamic programming whenever Eqs. (4.16)-(4.18) hold.*

We provide an informal proof of the above lemma which can easily be formalized. As stated before, Cormen et at. [118] have formally showed that an optimization problem

formulated using the Bellman equation satisfies the optimal substructure property can be solved by dynamic programming. Hence, to proof the lemma, we need only to show that whenever Eqs. (4.16)-(4.18) hold, the offloading problem as defined in Def. 4 shows an optimal substructure. This can be directly derived from the above examples that show that the optimal location vector for an EDT, with any node types, must include the optimal location vectors of its subtrees.

An example of $\leq_D$ that satisfy Eqs. (4.16)-(4.18) is one that employs the expected value of the distribution functions, i.e., $F_{Y_i}(x|\boldsymbol{l'}) \leq_D F_{Y_i}(x|\boldsymbol{l''}) \Leftrightarrow (E[Y_i|\boldsymbol{l'}] \leq E[Y_i|\boldsymbol{l''}])$.

We are now ready to use the dynamic programming based offloading scheme described in Algorithms 4-6. The first is the main algorithm that calls the latter two. Algorithm 5 visits the nodes of T in a bottom-up order. The optimal location for each visited node is obtained for each possible location of its parent node. Similarly, the optimal location for the parent node is decided for each of its parent's locations and so on until the algorithm reaches the root node. For each node, the optimal location is decided by comparing the CDFs resulting from locating the parent and child node on the mobile device and on the cloud. This information is stored for each node. Algorithm 6 traverses T starting from the root and visits all the nodes. At each node, the optimal location is selected depending on the already known location of the parent.

Algorithm 4: Solution algorithm

Data: T: EDT
Result: Location $\boldsymbol{l}* = [l_o, l_1, \cdots, l_n]$
/* Globals: $\boldsymbol{l}*$: optimal decision. $decision[r][l_c]$: optimal decision for v_r given it's parent location l_c. $cdf[r][l_c]$: optimal CDF for v_r given it's parent location l_c. $solved[r]$: marks if the node v_r has been solved/visited. */

```
1 begin
2 |  Solve(T);
3 |  BuildSolution(T, 0);
4 |  return l*;
5 end
```

The next section is dedicated to show how the application CDFs can be obtained.

4.4 Estimation of the Cumulative Distribution Function of the Application Costs

To estimate the cost of each module node, we employ a statistical method known as *bootstrap*. In this method, a random sampling with replacement is performed on the

Algorithm 5: Solve

```
   Data: T: EDT
1  begin
2      r ← T.root ;
3      if solved[r] then return;
4      solved[r] ← true;
5      foreach childTree ∈ r.children do
6          Solve (childTree);
7      end
8      foreach l_c ∈ {0,1} do
           /* l_c are the possible locations of v_r's parent                    */
9          cdf ← null;
10         l_r ← null;
11         foreach l'_r ∈ {0,1} do
12             cdf' ← GetCDF(r, l_c, l'_r);
13             if (cdf = null)|(cdf' ≤_D cdf) then
14                 cdf ← cdf';
15                 l_r ← l'_r;
16             end
17         end
18         decision[r][l_c] ← l_r;
19         cdf[r][l_c] ← cdf;
20     end
       /* where GetCDF(r, l_c, l_r) computes the CDF of node v_r according to Equations (4.9)
          to (4.13) depending on the type of v_r, i.e., leaf, series, probabilistic and
          parallel.  Note that the CDF of v_r's children were computed in line 6.    */
21 end
```

Algorithm 6: BuildSolution

```
   Data: T: EDT
   Data: l_c: Caller's location
1  begin
2      r ← T.root ;
3      l_r ← decision[r][l_c];
4      l*[r] ← l_r;
5      foreach childTree ∈ Children(r) do
6          BuildSolution (childTree, l_r);
7      end
8  end
```

profiled costs. This step obtains the needed observations of the random variables X_i^M, X_i^C and X_i^T. The module cost X_i, is then computed using Eq. (4.1). Next, measurements for the random variable of the cost of every *dependency node* are computed according to Eqs. (4.4)-(4.6) in a bottom-up approach up to the root. At the end, the values obtained for the root represent the needed samples to construct the cost of the application. This process is performed multiple times to get a statistically representative estimate of the cost CDF.

Depending on the offloading objective, multiple statistical measurements of the cost can be derived using this method. Examples of these measurements are the mean, confidence intervals, percentiles and standard deviation. This technique allows an efficient and accurate estimation of the running costs of the application under different scenarios.

Next, we list some scenarios where this estimation method is particularly useful in providing an accurate offloading decision that can accommodate the effects of the characteristics of the device and the network connecting the user to the cloud.

- *Application analysis for different network conditions.* The impact of the network on the performance of the application can be easily analyzed. Intuitively, when higher data rates are available, offloading the application modules becomes faster and may not add a significant cost, in terms of the response time, to the application execution. However, below a certain rate threshold, executing the application on the device might be faster. This threshold differs significantly from one application to the other and can be statistically estimated using the developed cost model.

- *Application analysis for different users behaviours.* Different users interact differently with the same application; for example, some users may only repetitively use specific features such as a given set of image processing operations on the full-resolution image. Other users, for example, may prefer a preview of multiple operations on a reduced-resolution image. Modifying the probabilities on the branching conditions for the EDT to match the user behaviour increases the precision in estimating the application costs.

- *Analysis of the application using different devices.* This step makes it possible to predict the performance of the application on multiple devices. Furthermore, the application can even be profiled for the same device with different configurations (e.g., energy savings and fully powered modes). Widely used applications can be also profiled using newly developed devices, or for new operating system upgrades. This may provide additional insights for the application developers and device manufacturers.

- *Analysis of the application under different application configurations.* Applications usually have multiple configuration parameters such as multiple graphic options in video

games, including texture resolution, use of shadows and number of polygons. Clearly, these settings affect the costs of running the modules of the applications and thus impact the execution cost of the application. These applications may easily have a large number of possible configurations, but clearly these may not necessary affect the cost of the application modules. The unique ability of proposed module to accommodate changes in the costs of individual modules without having to repeat the profiling of the entire application lends the model to be a ideal solution to profiling this genre of applications.

- *Analysis of the application with different design choices.* During application design, developers may have multiple design choices. The costs of running the application can be estimated for each candidate design and used as a tool to aid the developers in their choices.

In general, these use case scenarios can be performed using various statistical measurements (e.g., mean, standard deviation, quartiles and percentiles) as well as for diverse optimization objectives (e.g., energy and time minimization).

4.5 Performance Evaluation

We first analyze the accuracy of the cost model for each node type. We then evaluate the impact of the number of nodes of the application and the standard deviation of the cost of the modules on the accuracy of estimated values. In our experiments, we use the execution time as the target cost whereby the evaluation is carried out by comparing this average cost obtained through the measured time for running defined applications versus the average time estimated by the statistics-based cost model after profiling the applications.

4.5.1 Model and Cost Estimation Accuracy

We first evaluate the accuracy of the proposed cost model with respect to the types of dependency nodes in the application as well as the number of its modules.

Effects of the application size

Figure 4.4 shows the generic EDT used in the experiments where we vary the type of the dependency node v_0 as well as the number of its children modules. Each of the child module

nodes consists of several iterations of multiple random number computations and mathematical operations and memory reads and writes. Since we are interested in evaluating the accuracy of the model, we first assume that the offloading decision is to execute the application on the device. We run the experiments on an emulated device over a desktop computer with a 4 cores CPU and 12 GB of RAM. In all the experiments, we use the application completion time as the cost to be minimized. We executed each experiment 20 times while varying the number of children from 1 to 20. we profiled the applications obtaining observations for the module cost of each node. These costs are then used to estimate the cost using the proposed statistics-based cost model. Figure 4.5 depicts the average of the measured execution time as well as the average cost estimated by the model as obtained from profiling the individual modules using the bootstrap method. We use the *mean absolute percentage error (MAPE)* [119] to measure the accuracy of the model. It is defined as the means of the percentage error between each measured and estimated value. It is equal to $\frac{1}{N}\sum_{t=1}^{N}|\frac{M_t-E_t}{M_t}|$, where M_t and E_t are measured and estimated values, respectively, and N is the number of experiment runs.

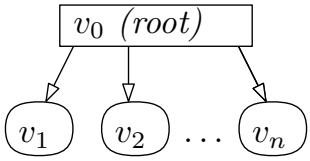


Figure 4.4: Execution Dependency Tree of the applications used to evaluate the statistics-based cost model.

For *series* and *probabilistic* dependency nodes, the MAPE value was 0.14% and 0.23%, respectively. We observed that increasing the number of nodes did not affect the error. On the other hand for the *parallel* root node, the MAPE value started increasing as the number of nodes exceeded 4. We attribute this increase to the hardware limitation of the hosting device that has 4 CPU cores. Since the device can only host 4 concurrent modules, true parallel execution cannot be achieved for more than 4 modules for the application. Hence, the actual application execution time becomes larger than the estimated one by the module. We plan to address this problem as a future work where we envision that the effects of the number of cores can be accounted for by using additional probabilistic dependency and series nodes in the EDT.

Effects of the standard deviation of the module costs

In this experiment, we maintained the same settings of the previous experiments, while modifying various operations within the child modules in the application shown in Figure

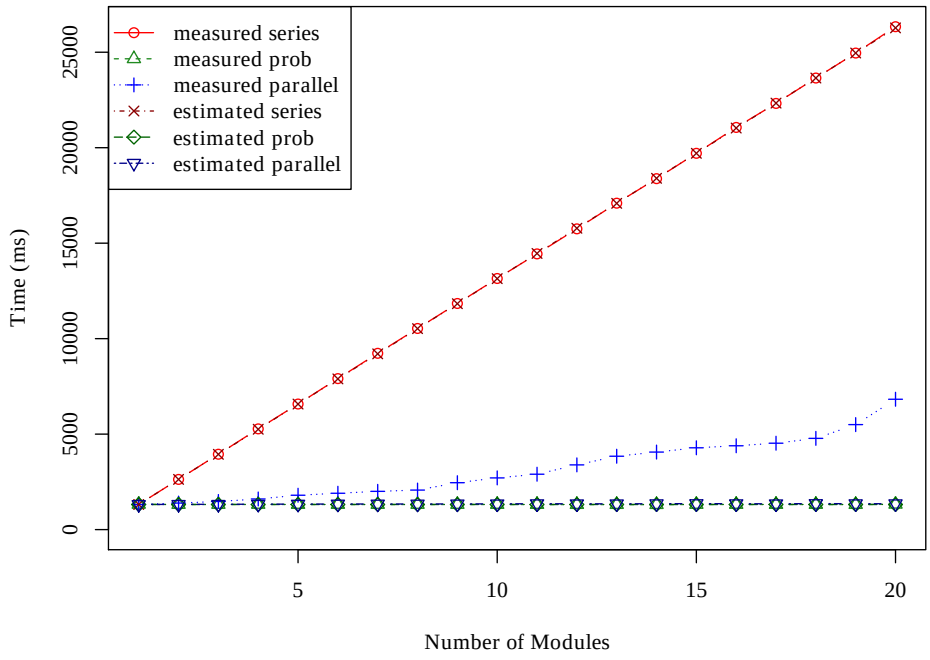


Figure 4.5: Measured and estimated execution cost versus the number of application modules.

4.4 in order to increase the variance in their costs. More precisely, we modified the different modules in order to obtain an estimated cost with a mean 7850 ms and standard deviation that varied between 0 ms to 2000 ms. We then evaluated the accuracy of the model for that range of the standard deviation. Figure 4.6 depicts the obtained results for both the measured and the model estimated costs.

For the applications with the *series*, *probabilistic* and *parallel* root nodes, the MAPE value was 2.08 %, 3.54 % and 7.23 %, respectively. As we expected the highest value was for the parallel node as was the case in the previous experiment. However, we noticed that increasing the standard deviation did not significantly affect the results for the three applications.

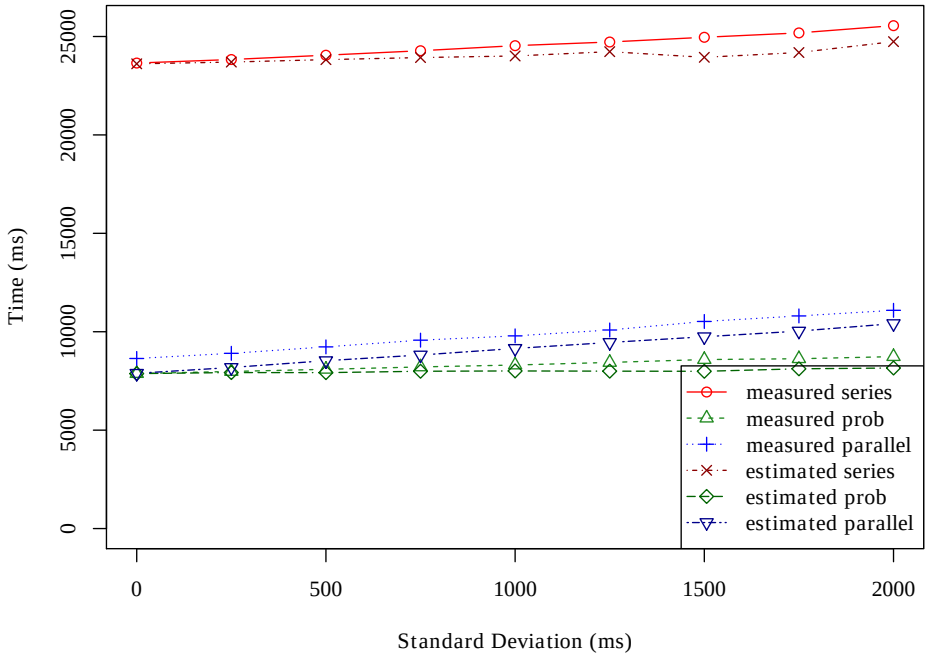


Figure 4.6: Average measured and estimated cost varying the standard deviation of module costs.

4.5.2 Accuracy of the Offloading Algorithm

In this experiment, we evaluate the efficiency of the offloading algorithm using the face-detection application presented in Section 3.5 and whose EDT is depicted in Figure 3.3. Using the application, the user either executes the face detection method (fd0) directly or

chooses to execute a thumbnail face detection operation (method th). The latter method first calls the image resizing operation (method rz) then the face detection one (fd1). We developed a profiler as part of the offloading framework. The EDT is first built automatically by observing the execution paths when profiling the application. At this stage, the probability of running each child of the probabilistic dependency node is also calculated from these runs. Next, the profiler samples the cost of each module from several runs of the application. Figure 3.3 also shows the sampled average cost values for the modules. For different conditions or changes in the state of the environment, such as a different mobile device or network bandwidth, the EDT must be updated and the offloading algorithm must be run to obtain the new optimal offloading decision. This is similar to other works like MAUI [2]. The mobile application runs on a Microsoft Phone Windows Simulator, and the offloading server is a .Net application running on Ubuntu 14.04 with Mono Framework [120] on a DigitalOcean droplet [121]. On the server, we control the data rate of the communication between the mobile application and the offloading server in order to carry on the simulations.

As described in Section 4.3, the proposed dynamic programming algorithm calculates the cost of different offloading choices for every module and eventually for all the location vectors as it traverses the tree. Using these costs, we can estimate the costs of various location vectors without running the application. To measure the accuracy of the algorithm in calculating these costs, Figure 4.7 plots the estimated (marked as measured on the plot) costs for three offloading decisions vectors: no offloading, offloading the entire application and employing the developed algorithm to obtain the best location vector while varying the available data rate of the underlying network. The plot using the simulated device that is connected to the cloud for these three scenarios.

Clearly, the offloading decision as obtained by the proposed algorithm shows the lowest costs for different network data rates. On the other hand, the MAPE values were 7.3%, 4.7% and 4.63% for the no offloading, offloading the entire application and employing the developed algorithm scenarios, respectively. The slight increase in these values compared to those computed in the previous experiments can be explained by noting that the cost model does not include some additional overhead that is incurred in actual execution environments. Our investigations showed that this overhead is the result of executing the offloading framework in addition to time taken for, TCP session establishment, resource allocation and memory access, on the device. Noting that for this experiment, we did not use the user device to profile the application. This result suggests that future additional work is needed to include these effects in the cost models in order to improve the precision of the model.

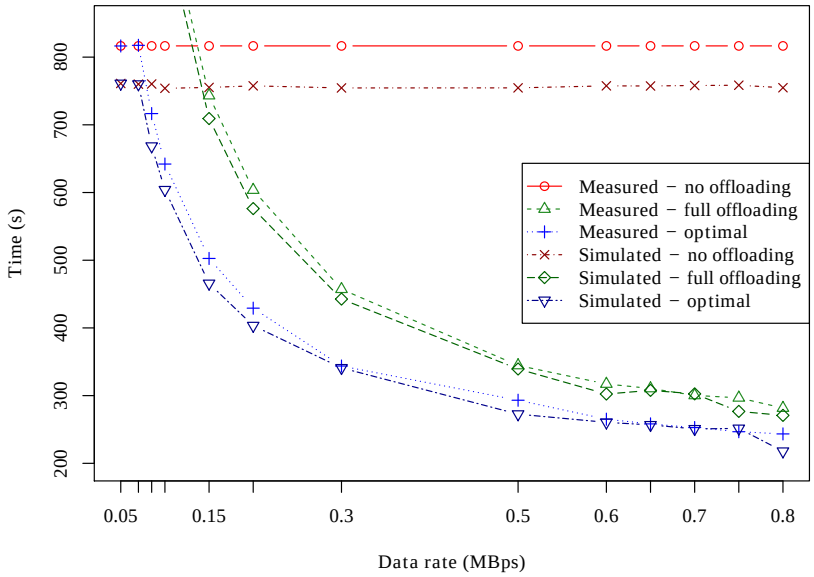


Figure 4.7: A comparison of the FD application execution cost, simulated and measured, under the offloading decision from the proposed algorithm and full offloading and no offloading decisions, while varying the network data rate.

4.5.3 Cost Model Sensitivity to Varying Contexts

Next, we illustrate the use of the developed statistics-based cost model to accurately measure the performance of developed applications under different scenarios including diverse user preferences, different device configurations and when using various network technologies. With the wide prevalence of remote and frequent updates of mobile applications, we envision that the new model can serve as an important application analysis tool for application developers. While modelling these scenarios, the developed tool can produce histograms, density functions and CDFs that accurately reflect the varying costs of the application. In addition, statistical measurements can also be computed, such as the quartiles, expected values and the standard deviation, in order to aid the developers in perfecting the applications performance. To this end, the following sections demonstrate some of these scenarios.

User Behaviour

Clearly, users interact differently with their applications. They may also exhibit a repetitive pattern while using certain applications. For example, for a face-detection application, one

$P(th)$	1st Q	Median	Mean	3rd Q	95 %	σ
0	428.1	468.7	467.5	510.9	550.35	58.74
0.25	111.80	443.90	370.00	500.60	543.95	177.95
0.5	75.10	111.80	269.80	468.40	536.85	199.42
0.75	68.72	83.10	173.40	111.90	525.27	172.98
1	65.60	75.18	75.37	86.50	94.26	13.61

Table 4.1: A sample of statistical measurements for the FD application for different users behaviours (σ represents the standard deviation).

user may be used to execute the application directly on full sized images. On the other hand, another user may almost allows prefer to resize all images to their thumbnails before executing the face-detection application. Different users behaviours may as well fall in between these two behaviours. Figure 4.8 plots the histograms depicting the cost, in terms of the execution time for different users of the face-detection application as they use a face-detection for a sample of 20 images. In the figure, each histogram depicts a different user behaviour as marked by the parameter $P(th)$. This parameter represents the probability that the user will use the face-detection for a thumbnail of the image instead of the full image. The top left histogram with $P(th) = 0$ depicts the cost for a user that never uses the thumbnail preview. In this case, the histogram shows that the user will almost always experience a long execution time for the application that ranges between 400 s and 600 s. As we see users with a higher $P(th)$, their histograms show smaller and smaller average wait times. For the user that almost always uses thumbnail images face-detection, the wait is minimal and falls within the range of 60 s to 100 s. Figs. 4.9 and 4.10 plot the density functions of the total cost of the face-detection application corresponding to the histograms and their CDFs, respectively. Additionally, Tbl. 4.1 provides a sample of some common statistics for the application cost that can be useful to application developers. The table shows that the lowest mean for the cost when the user always uses the thumbnails $P(th) = 1$. It also shows that the variance in this case is still small with a standard deviation of 13.61 s. The second to last column shows the 95 cost percentile. It indicates that 95 % of the executions for that user will be faster than 94.26 s.

Cost sensitivity to different device and network configurations

Second, we demonstrate the use of this model when the developer is testing the behaviour of the applications as it runs on several mobile devices that use different networks. More precisely, we consider two devices one of which is one and a half times faster. We mark the first as a device with a speed up factor of 1 while the other with a 1.5 speed up factor.

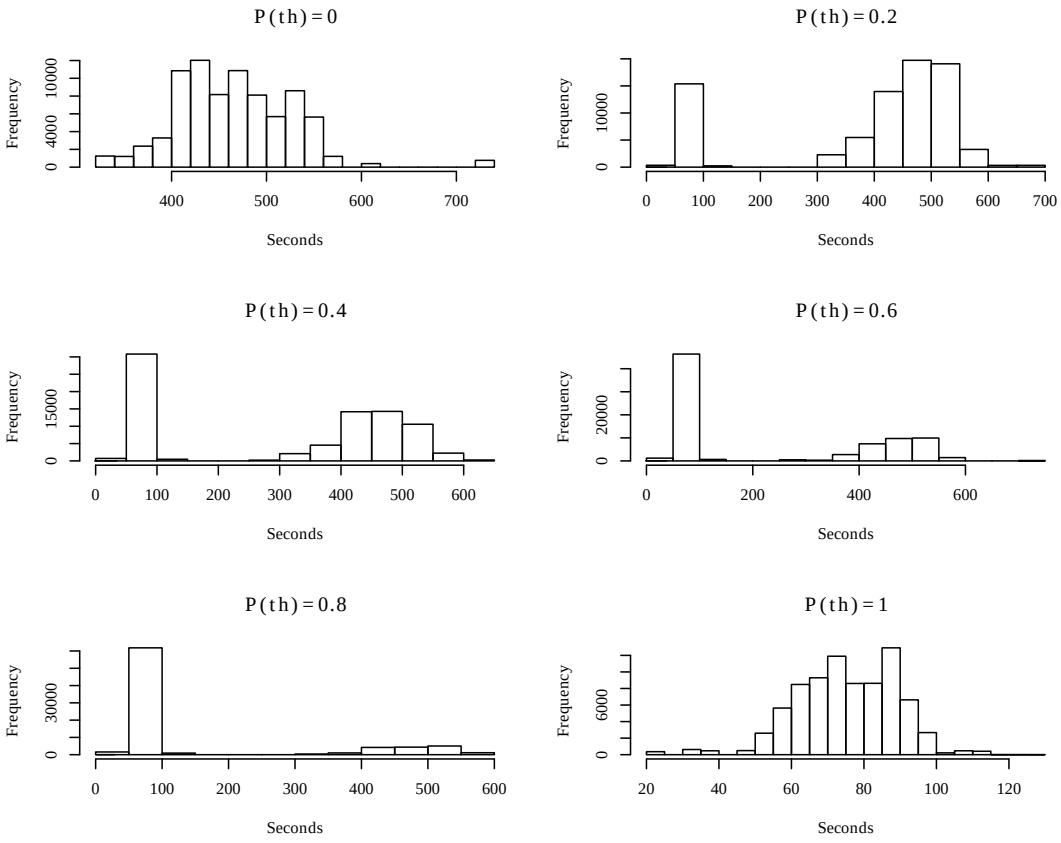


Figure 4.8: Histograms of the estimated execution time of the FD application for different users behaviours.

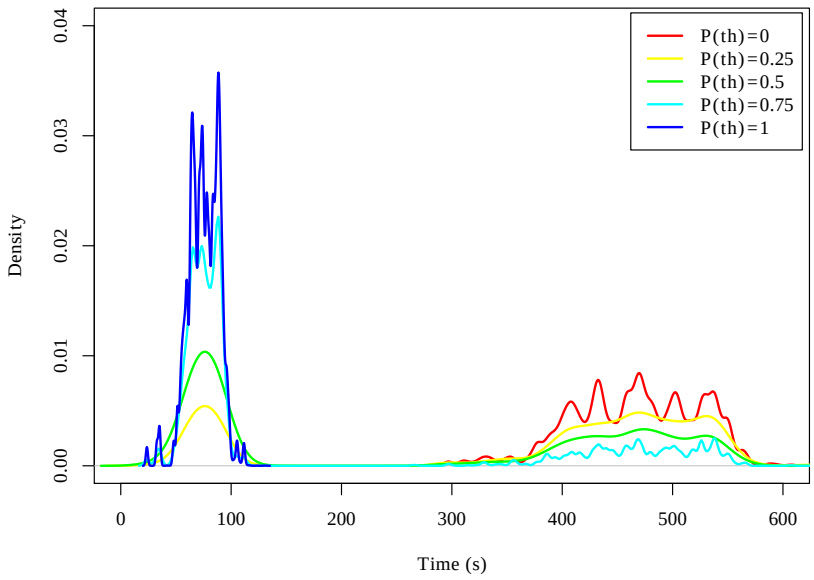


Figure 4.9: Density of the estimated execution cost of the FD application for various users behaviours.

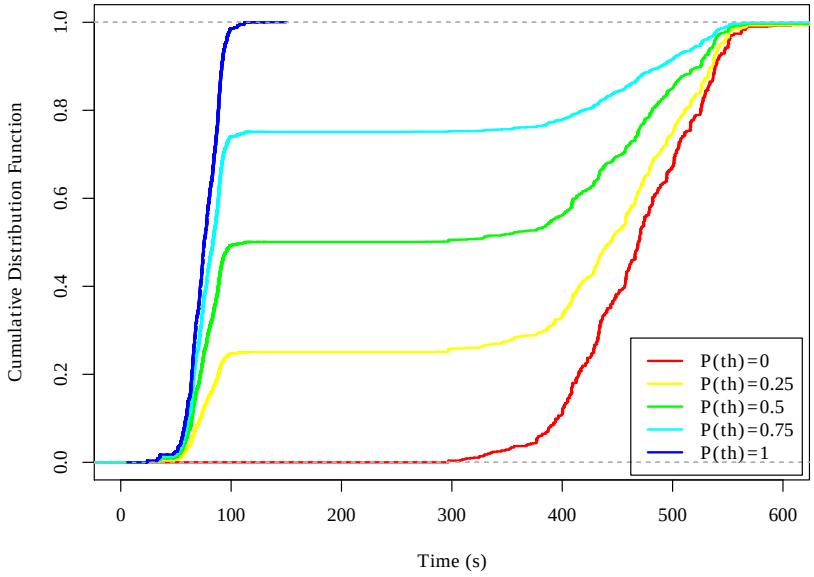


Figure 4.10: Estimated cost CDF for the FD application for various users behaviours.

Here, the *speed up factor* is a simple measure of the CPU speed of the mobile device that allows comparing the relative speed of different devices. Each device experiences different network conditions with varying data rates. We profile the same face-detection application as it runs on these two devices while considering three different offloading decision, namely, execute the application entirely on the device, remotely on the cloud or using the decision resulting from our offloading decision algorithm that only offloads the execution of the face-detection operation on full size images (marked by the module fd0). These three decisions are referred to as, No-offload, Full-offload and fd0-offload, respectively, in the generated figures.

Figure 4.11 depicts the expected costs, in terms of the execution times, for the two devices while varying the available data rate for the underlying network. Clearly, increasing the device's computational power has a considerable impact when the application runs on the mobile, while for the both the partial and full offloading solutions the benefit is minimal. Such an analysis may help the developer in quantifying the real impact of the mobile device characteristics on the application performance while considering various offloading solutions for users using different networking technologies.

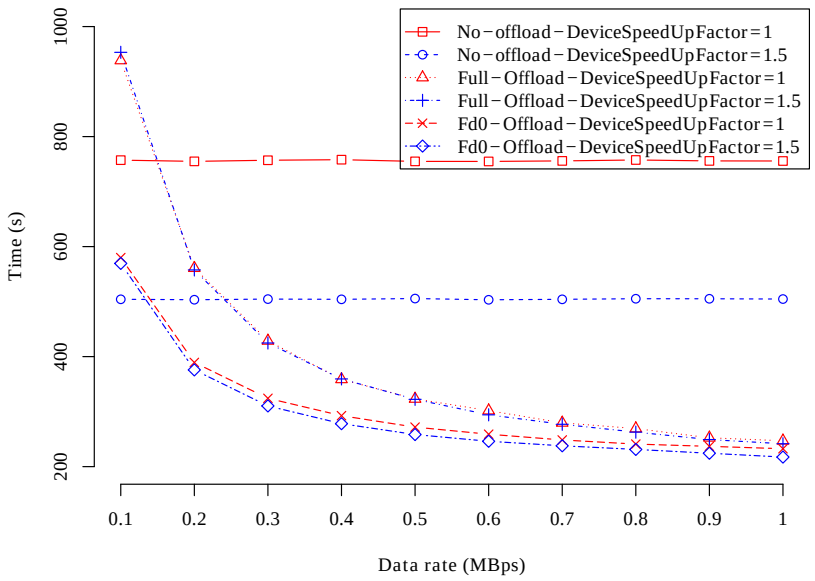


Figure 4.11: Expected execution time of the FD application for different devices and offloading decisions and network conditions.

Cost sensitivity to Users behaviors and network conditions

Another useful use case scenario for the developed tool is when the developer is interested in investigating the dependencies among several parameters given a preselected offloading decision vector for the application. This can help the developer in identifying the best scenarios for adopting a certain offloading decision for instance. Figure 4.12 demonstrates, such an example where it plots the estimated average cost as the network data rates and the user behaviour (defined by the probability of selecting a thumbnail image $P(th)$) are varied for a location decision vector that uses the cloud only Whenever a user chooses to employ face detection on full size images. As shown in the figure, for low network speeds, the user behaviour has a significant impact on application's running time. As the data rate increase, the faster the application runs even for those users that choose to use full size images.

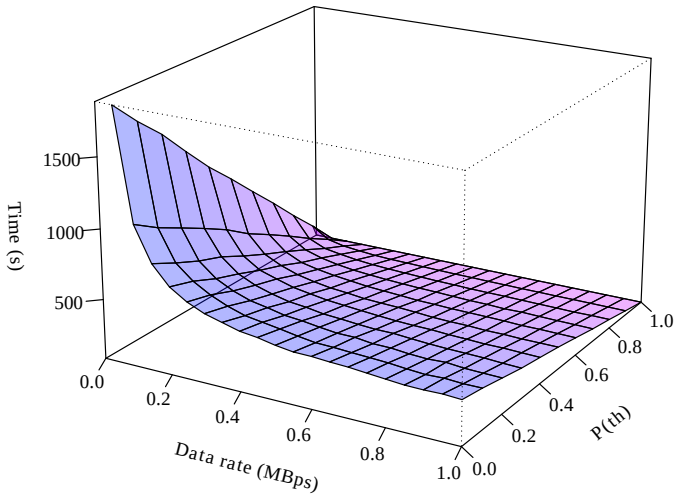


Figure 4.12: Execution cost versus the network data rate and users behaviours when only fd0 is offloaded.

Another similar test example is shown in Figure 4.13. The figure shows the average application execution time for different data rates and device speed up factors when face-detection for full images is done in the cloud. Clearly, for this offloading decision, the impact of the device speed up is negligible.

Similarly, Figure 4.14 shows the cost for the face-detection application under different

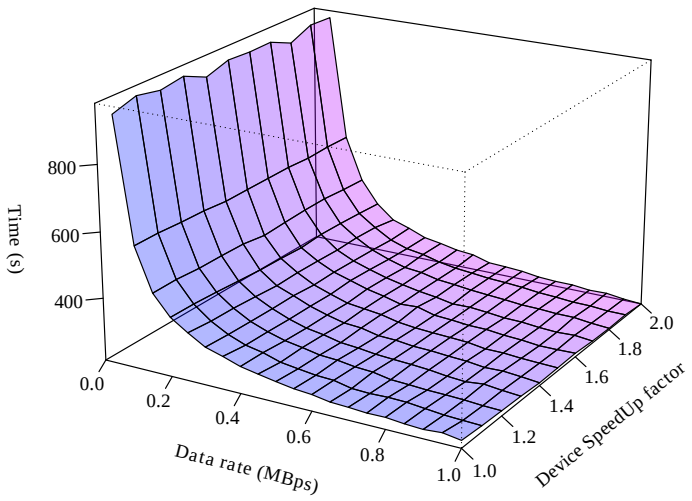


Figure 4.13: Execution cost versus the network data rate and device speed ups when only fd0 is offloaded.

user behaviours and device speed factors and no offloading. For users that perform face-detection on the full image, the speed of the device has a significant impact and the application running time can be reduced by about 50 % with a speed factor of 2. For users that are frequently use thumbnail face-detection, reflected by a larger value for $P(th)$, the benefit of a faster device is smaller.

4.6 Summary

This chapter described a novel cost model that employs the EDT application model and where costs are represented by random variables and their CDFs rather than the commonly used fixed cost averages. Using these CDFs and the structure of the tree, the CDF of the entire application is estimated for various factors that affect the offloading decision. These factors include the users behaviours while interacting with the application as well as the underlying network conditions. Using this cost model, a novel offloading algorithm was also introduced to optimally select which components of the application can be offloaded to the cloud in order to optimize a given objective. Finally, we describe several use case scenarios where the developed model can be used as an efficient application profiling tool by the developers to continuously enhance the performance of the application given different

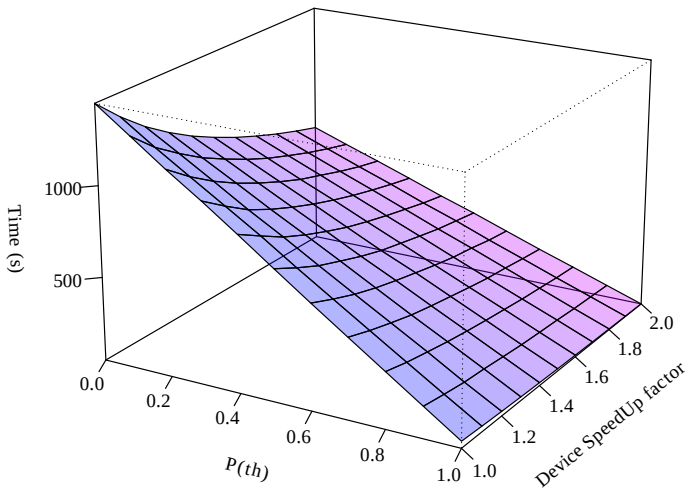


Figure 4.14: Execution cost versus device speed ups and users behaviours when no modules are offloaded.

network and device configurations as well as users behaviours.

Chapter 5

Application Offloading via Robust Optimization

In some scenarios, we only have partial information about the application costs. This occurs, for example, when we have not yet gathered enough data from profiling for a given user or when the parameters, such as network data rate, fluctuates rapidly. In this chapter, we propose a novel robust formulation of the problem using a novel interval-based cost model and the EDT application model.

5.1 Interval-based Cost Model

The cost model uses the EDT application model proposed in Chapter 3. Similar to the other cost models, three random variables model the three costs for each application module, i.e., execution cost in the mobile and on the cloud, and the transmission cost. This results in $k = 3n$ random variables where n is the number of application modules. These costs can represent time, energy, monetary costs, etc. For ease of presentation, we summary the notation used in this chapter in Table 5.1.

Notation	Description
n	number of modules in the application
$k = 3n$	number of variables
$p(i)$	parent of module i
ρ_i	probability of execution of module i in one execution of the application
α_i	average execution cost on the mobile device for node i
β_i	average execution cost on the cloud for node i
ζ_i	average transmission cost for node i
δ_{α_i}	additional execution cost, from uncertainty, on the mobile device for node i
δ_{β_i}	additional execution cost, from uncertainty, on the cloud for node i
δ_{ζ_i}	additional transmission cost, from uncertainty, for node i
R_c	set of modules that are restricted to run in the cloud
R_m	set of modules that are restricted to run on the mobile device
F	total cost of the application: $F = F' + F''$
F'	expected application cost
F''	expected additional application cost from uncertainty
$\boldsymbol{c}$	vector of module's expected costs (execution and transmission) per application execution; it includes ρ_i, α_i, β_i and ζ_i
$\boldsymbol{d}$	vector of module's expected additional costs (execution and transmission) from uncertainty per application execution; it includes ρ_i, δ_{α_i}, δ_{β_i} and δ_{ζ_i}
Γ	robust coefficient, i.e., number of costs that may take the worst case value
	Decision Variables
$\boldsymbol{y}$	vector of variables of the application cost function; it includes $\boldsymbol{x}$, $\hat{\boldsymbol{x}}$ and $\boldsymbol{T}$
$\boldsymbol{x}$	vector of offloading decisions; x_i is the offloading decision for module i
$\hat{\boldsymbol{x}}$	vector of inverse offloading decisions; $\hat{x_i} = 1 - x_i$ is the inverse of the offloading decision for module i
$\boldsymbol{T}$	vector of T_i; $T_i = \lvert x_i - x_{p(i)} \rvert$ denotes if module i and its parent run in different contexts (mobile device and cloud)

Table 5.1: Summary of notation of the chapter

Let $\tilde{\alpha}_i$ and $\tilde{\beta}_i$ be random variables that model the execution cost of component v_i, $i \in \{1, \ldots, n\}$, in the cloud and in the mobile, respectively, and let $\tilde{\zeta}_i$ be the transmission costs of component v_i. Each random variable is defined within an interval, referred to as *robustness interval*, i.e., $\tilde{\alpha}_i \in [\alpha_i, \alpha_i + \delta_{\alpha_i}]$, $\tilde{\beta}_i \in [\beta_i, \beta_i + \delta_{\beta_i}]$ and $\tilde{\zeta}_i \in [\zeta_i, \zeta_i + \delta_{\zeta_i}]$, where $\alpha_i, \delta_{\alpha_i}, \beta_i, \delta_{\beta_i}, \zeta_i, \delta_{\zeta_i} \geq 0$. For example, as we will see next, if $\tilde{\alpha}_i \in [3, 3 + 2]$, we may say that the execution cost in the cloud for node v_i is 3 s on average, and 5 s in the worst case.

There are multiple ways of choosing the interval values for each cost variable. For example, considering the execution cost in the cloud, a simple approach is to choose α_i as the mean value of observed costs, and set $\alpha_i + \delta_{\alpha_i}$ as the maximum of the observed costs. Another approach is to set $\delta_{\alpha_i} = \epsilon_{\alpha_i}\alpha_i$ where $\epsilon_{\alpha_i} \geq 0$ is a measurement error when profiling the cost. Also, if we know (or can estimate) the CDF of the random variable $\tilde{\alpha}_i$, we can use statistical measurements, for example, $\alpha_i = \text{mean}(\tilde{\alpha}_i)$ and $\alpha_i + \delta_{\alpha_i}$ equals to the $90th$ percentile of $\tilde{\alpha}_i$. The goal of the application offloading problem must be considered when computing the robustness intervals. For instance, if the goal is to minimize the application cost when the module's cost take the worst case, for example for applications with strict deadlines, we must use the first example.

From the EDT model, we know that each module node has a probability to run on any execution of the application. For example, in a photo editing application, a module that crops the image will run in some occasions only, according to the needs or preferences of the user. Let ρ_i be the probability that v_i runs in any execution of the application. Value ρ_i is computed from the probabilities in the EDT as follows. If v_1 is the root of the tree, then $\rho_1 = 1$, otherwise, the p_i is equal to the execution probability of v_i's parent node multiplied by the probability that v_i runs given its parent is running, which is specified by the dependency nodes.

In addition to module costs and execution probabilities, the offloading problem must consider location restrictions of the modules. For example, a module that accesses the microphone of the mobile device to capture commands spoken by the user is restricted to run on the mobile. On the contrary, a module that performs speech recognition on the captured audio is restricted to run on the cloud whenever the mobile device does not have the needed computational resources or software for speech recognition. Let $R_c \subseteq \{1, \ldots, n\}$ and $R_m \subseteq \{1, \ldots, n\}$ be the sets of modules that are restricted to run in the cloud and on the mobile device, respectively. Clearly, $R_c \cap R_m = \emptyset$.

The offloading algorithm must make an offloading decision for each module of the application. Let x_i be the offloading decision of node v_i where decision $x_i = 0$ ($x_i = 1$) means that module v_i runs in the mobile (cloud). Also, let $p(i)$ be the parent node of v_i.

Then, $x_{p(i)}$ is the location of parent of node i in the EDT tree. By definition, node v_1 is the root of the tree and $x_1 = 0$ meaning that the application starts in the mobile.

Definition 5. *The expected worst case cost F of running the entire application, given the modules' interval-based costs α_i, β_i, ζ_i, δ_{α_i}, δ_{β_i}, δ_{ζ_i}, module's execution probabilities ρ_i, and offloading decision vector $\boldsymbol{x}$, is given by:*

$$F = F' + F'', \tag{5.1}$$

where

$$F' = \sum_{i=1}^{n} \rho_i \alpha_i x_i + \sum_{i=1}^{n} \rho_i \beta_i \hat{x}_i + \sum_{i=1}^{n} \rho_i \zeta_i T_i, \tag{5.2}$$

$$F'' = \sum_{i=1}^{n} \rho_i \delta_{\alpha_i} x_i + \sum_{i=1}^{n} \rho_i \delta_{\beta_i} \hat{x}_i + \sum_{i=1}^{n} \rho_i \delta_{\zeta_i} T_i, \tag{5.3}$$

and $\hat{x}_i = 1 - x_i$ and $T_i = |x_i - x_{p(i)}|$.

Function F' denotes the expected application cost and is computed in Eq. (5.2) as the sum of expected cost of each module. The expected cost of a module v_i the sum of the expected cost of the module on the cloud and on the mobile, and the expected transmission cost, multiplied by the probability ρ_i that the module will run. This value does not account for uncertainty in the parameters and thus, in robust terminology, the problem of optimizing F' is referred to as the *nominal problem*.

However, we know that the application cost is uncertain, and module costs may take values within the interval. Value F'' accounts for the additional cost due to uncertainty that is added to the nominal application. It is computed in Eq. (5.3) as the sum of the additional costs due to uncertainty from each module, multiplied by the probability of execution of the module.

Consequently, function F in Eq. (5.1) represents the *expected worst case* cost of running the entire application. We refer to this cost as *expected* worst case as the cost is a weighted sum based on the probability of execution of the modules. In the following, when we refer to *worst case cost*, we will be referring to *expected worst case* cost as defined in Equation (5.1).

To simplify the notation, we represent all the coefficients of Equations (5.2) and (5.3) through two vectors $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{R}^k_{\geq 0}$, and all the offloading decisions, i.e., x_i, $\hat{x}_i$ and T_i, by vector $\boldsymbol{y} \in \{0,1\}^k$, $k = 3n$. Then, the cost of the application is expressed by

$$F = F' + F'' \tag{5.4}$$

where

$$F' = \sum_{j=1}^{k} c_j y_j, \tag{5.5}$$

$$F'' = \sum_{j=1}^{k} d_j y_j, \tag{5.6}$$

and

$$\boldsymbol{c} = \begin{bmatrix} \rho_1\alpha_1 \\ \vdots \\ \rho_n\alpha_n \\ \rho_1\beta_1 \\ \vdots \\ \rho_n\beta_n \\ \rho_1\zeta_1 \\ \vdots \\ \rho_n\zeta_n \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} \rho_1\delta_{\alpha_1} \\ \vdots \\ \rho_n\delta_{\alpha_n} \\ \rho_1\delta_{\beta_1} \\ \vdots \\ \rho_n\delta_{\beta_n} \\ \rho_1\delta_{\zeta_1} \\ \vdots \\ \rho_n\delta_{\zeta_n} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ \hat{x}_1 \\ \vdots \\ \hat{x}_n \\ T_1 \\ \vdots \\ T_n \end{bmatrix}. \tag{5.7}$$

We shall now present the problem formulation of the application offloading problem based on this cost model.

5.2 Robust Problem Formulation

Given the cost definition above, we could state the offloading problem for given cost intervals as to find the location vector $\boldsymbol{y}$ that minimizes F in Equation (5.4). This formulation could be rephrased as to find the offloading decision that guarantees a minimum worst case application cost for all realizations of the costs within the intervals. This problem formulation is, in fact, the traditional robust optimization objective. The formulation, however, had been criticized in the past for being too conservative.

Traditional robust optimization aims at optimizing the worst case cost that may arise due to uncertainty, which is done at the expense of higher nominal cost. For our problem,

the nominal cost refers to the expected application cost, which is computed by Equation (5.6). But, for scenarios where it is unlikely that all parameters will take the worst case simultaneously, this formulation is considered too conservative. Therefore, similar to works such as [103], we present a problem formulation that addresses the uncertainty of the costs while addressing said conservatism.

We present a robust formulation of the application offloading problem where a subset of the k may have uncertainty simultaneously, and not necessarily of all of them. If a cost parameter is deterministic, then its value is certain, otherwise, it is allowed to take values within its interval. For example, $\tilde{\alpha_i} = \alpha_i$ if $\tilde{\alpha_i}$ is deterministic, and $\tilde{\alpha_i} \in [\alpha_i, \alpha_i + \delta_{\alpha_i}]$ if $\tilde{\alpha_i}$ is uncertain. This approach, also employed by Bertsimas and Sim in [103], addresses the pessimistic nature of traditional formulations of the robust optimization problem. Furthermore, this formulation covers all cases ranging from the non-robust problem where no coefficient may change, to the traditional scenario where all the coefficients change. Let $\Gamma \in [0, k]$ be the number of parameters that may take worst case cost simultaneously.

Definition 6. *We define the application offloading problem, given* $\boldsymbol{c}$, $\boldsymbol{d}$ *and* Γ, *as finding a solution* $\boldsymbol{y} \in Y$ *that minimizes the maximum application cost* F^* *for all* $S \subseteq \{1, \ldots, k\}$, *where* $|S| \leq \Gamma$. $\Gamma \in \{0, \ldots, k\}$ *is the number of parameters that are uncertain and* S *is the set of parameters that are uncertain. Formally,*

$$\begin{aligned} F^* = \min_{\boldsymbol{y}} \quad & \max_{\forall S \subseteq \{1,\ldots,k\}, |S| \leq \Gamma} \left(\sum_{j=1}^{k} c_j y_j + \sum_{j \in S} d_j y_j \right) \\ s.t. \quad & \boldsymbol{y} \in Y, \end{aligned} \tag{5.8}$$

$$Y = \{\boldsymbol{y} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \\ \boldsymbol{T} \end{bmatrix} \in \{0,1\}^k, \boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{T} \in \{0,1\}^n, k = 3n|$$

$$x_i + \hat{x}_i = 1, \forall i \in \{1, \ldots, n\} \tag{5.9}$$

$$x_i - x_{p(i)} - T_i \leq 0, \forall i \in \{1, \ldots, n\} \tag{5.10}$$

$$-x_i + x_{p(i)} - T_i \leq 0, \forall i \in \{1, \ldots, n\} \tag{5.11}$$

$$x_{p(1)} = 0, \tag{5.12}$$

$$x_j = 1, x'_j = 0, \forall j \in R_c \tag{5.13}$$

$$x_j = 0, x'_j = 1, \forall j \in R_m\} \tag{5.14}$$

where $\boldsymbol{c}$, $\boldsymbol{d}$ *are defined in Eq.* (5.7).

In other words, we want to find an offloading decision vector $\boldsymbol{y}$ such that the maximum application cost for all set S of parameters, $|S| \leq \Gamma$, is minimum.

Constraint (5.9) specifies that a module runs either on the mobile or the cloud. Constraints (5.10) and (5.11) guarantee that $T_i = |x_i - x_{p(i)}|$, i.e., that there is a transmission cost if a module and its parent run in different locations (mobile device or cloud). Equation (5.12) means that, by definition, the application starts in the mobile device, i.e., the parent of the first module to run runs in the mobile device. The constraints (5.13) and (5.14) specify that some modules are restricted to run in the cloud or in the mobile device, respectively.

Our robust formulation is flexible as it is configured by specifying appropriate robustness intervals and number Γ. In general, a higher value of Γ increases the level of robustness of the offloading decision at the expense of higher application cost. There are two special cases. On the one hand, if $\Gamma = 0$, then the robust problem is equivalent to the non-robust application offloading problem for given vector of costs $\boldsymbol{c}$. On the other hand, if $\Gamma = k$, then the robust problem is equivalent to minimizing the application cost when all the modules exhibit the worst case cost.

5.3 Robust Offloading Decision Algorithm

Contrary to the first impression which suggests that Problem (5.8) must be solved for an exponential number of sets S, the problem can be efficiently solved by solving no more than $k+1$ BLP problems. In fact, Problem (5.8) is a Robust Combinatorial Optimization problem which is studied by Bertsimas and Sim in [103]. Let $\boldsymbol{d} = [d_1, \ldots, d_k]$ as defined in Eq. (5.7). Without loss of generality, let $d_1 \geq d_2 \geq \ldots \geq d_k$, and let us define $d_{k+1} = 0$ for notational convenience. The authors prove in Theorem 3 in [103] that, when $d_1 \geq d_2 \geq \ldots \geq d_k$, Problem (5.8) is equivalent to:

$$
\begin{aligned}
F^* = \min_{\boldsymbol{y}} \Bigg[& \Gamma * d_1 + \min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} + \sum_{j=1}^{1} (d_j - d_1) y_j \right), \\
& \Gamma * d_2 + \min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} + \sum_{j=1}^{2} (d_j - d_2) y_j \right), \ldots, \\
& \Gamma * d_{k+1} + \min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} + \sum_{j=1}^{k+1} (d_j - d_{k+1}) y_j \right) \Bigg] \\
= \min_{\boldsymbol{y}} \Bigg[& \Gamma * d_1 + \min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} \right), \ldots, \\
& \Gamma * d_l + \min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} + \sum_{j=1}^{l} (d_j - d_l) y_j \right), \ldots, \\
& \min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} + \sum_{j=1}^{k} d_j y_j \right) \Bigg].
\end{aligned}
\tag{5.15}
$$

The reader is referred to [103] where a proof and description of Eq. (5.15) is developed.

Following Eq. (5.15) and Algorithm A in [103], Algorithm 7 solves Problem (5.8) solving at most $k + 1 = 3n + 1$ nominal problems.

The first step of the algorithm is to reorder the coefficients so that $d_1 \geq d_2 \geq \ldots \geq d_k$. Therefore, we find a permutation function Π (Line 3) and apply it to vectors $\boldsymbol{d}$ and $\boldsymbol{c}$ (Lines 4 and 5). Next, we solve the $k + 1$ combinatorial optimization problems (Lines 8 to 16), finding $\hat{\boldsymbol{y}} \in Y$ that solves the problem for the reordered coefficients (Line 11). Thus, we obtain the optimal value $\boldsymbol{y}$ for the robust problem by permuting back the elements in vector $\hat{\boldsymbol{y}}$, having $\boldsymbol{y} = \Pi^{-1}(\hat{\boldsymbol{y}})$ (Line 15). Variable F^* holds the application cost for the optimal solution. Vector $\boldsymbol{y}$ is the solution of Problem (5.8). Finally, we obtain the offloading decisions of the n modules of the application by having the first n values of vector $\boldsymbol{y}$, i.e., $\boldsymbol{x}^* = (y_1, \ldots, y_n)$ (Line 16).

Note that Algorithm 7 preserves polynomial solvability of the nominal problem $\min_{\boldsymbol{y} \in Y} \left(\boldsymbol{c}^T \boldsymbol{y} \right)$, i.e., if the nominal problem is polynomially solvable, then the robust problem is also polynomially solvable. This is because the robust algorithm solves at most $f + 1$ nominal problems, where f is the number of distinct values of $d_1, \ldots, d_k$.

Algorithm 7 can be used in the same contexts as traditional application offloading algorithms. For instance, the algorithm can be run either on the mobile device or on the cloud, in which case the solution is communicated to the mobile device. The framework must periodically profile the parameters of the environment such as module costs and

Algorithm 7: Robust application offloading, solves Problem 5.8

Data: $\boldsymbol{c}$, $\boldsymbol{d}$: as defined in Eq. (5.7); $\Gamma \in [1, k]$: robustness coefficient
Result: $\boldsymbol{x}^*$: optimal robust offloading decision

begin
  $F^* \leftarrow \infty$; `// optimum application cost`
  `//` Let $\Pi : \Re^k \to \Re^k$ `be a permutation function such that` $\Pi(\boldsymbol{d})$ `is ordered`
  $\Pi \leftarrow$ find permutation function given $\boldsymbol{d}$;
  $\hat{\boldsymbol{d}} \leftarrow \Pi(\boldsymbol{d})$; `//` $\hat{d}_1 \geq \ldots \geq \hat{d}_k$
  $\hat{\boldsymbol{c}} \leftarrow \Pi(\boldsymbol{c})$;
  $k \leftarrow |c|$;
  $\hat{d}_{k+1} = 0$;
  foreach $l \leftarrow 1$ **to** $k + 1$ **do**
    if $l > 1$ **and** $\hat{d}_l = \hat{d}_{l-1}$ **then**
      continue; `// to next iteration`
    $\hat{\boldsymbol{y}} \leftarrow \arg\min_{\boldsymbol{y} \in Y} \left(\hat{\boldsymbol{c}}^T \boldsymbol{y} + \sum_{j=1}^{l} (\hat{d}_j - \hat{d}_l) y_j \right)$;
    $F \leftarrow \Gamma \hat{d}_l + \min_{\boldsymbol{y} \in Y} \left(\hat{\boldsymbol{c}}^T \boldsymbol{y} + \sum_{j=1}^{l} (\hat{d}_j - \hat{d}_l) y_j \right)$;
    if $F < F^*$ **then**
      $F^* \leftarrow F$;
      $\boldsymbol{y} \leftarrow \Pi^{-1}(\hat{\boldsymbol{y}})$; `// reorder solution vector with inverse of permutation` Π
      $\boldsymbol{x}^* \leftarrow (y_1, y_2, \ldots, y_n)$; `// optimal solution`

network data rate. The profiled data is used to compute the robust intervals presented above.

In the next Section, we study the performance of the robust solution compared to the non-robust counterpart for different scenarios.

5.4 Performance Evaluation

We evaluate our robust formulation with a FD application [14] and with randomly generated applications.

5.4.1 Configuration of applications

Table 5.2 shows a summary of the properties of the EDT of the applications, including the types of module dependencies, the number of module nodes, and the tree height.

	dependencies			
Application	series	prob.	module nodes	tree height
FD	✓	✓	5	4
Single-Series	✓		100	2

Table 5.2: Properties of the Execution Dependency Tree of the applications used for evaluating the interval-based offloading method.

Face Detection Application

The FD application is a Microsoft Windows Phone application that performs face detection using Accord.NET Framework [122]. Figure 5.1 shows the EDT of the application. The user can perform face detection on either thumbnails of images (node th) or full resolution (node fd0) with equal probability, i.e., $\rho_{\text{th}} = \rho_{\text{fd0}} = 0.5$. If thumbnail face detection is chosen, the image is first resized (node rz) and then face detection is done on the resized image (node fd1).

Figure 5.2 shows the boxplots of the execution costs of each module, on the mobile and on the cloud, as profiled from runs of the application for 200 images. The execution cost of all the modules except fd0 is close to 0 s on both the mobile and the cloud. For fd0, the execution cost on the mobile varies from 10 s to 18 s with a mean of 15.59 s, while on the cloud the cost has a mean of 71.85 s and takes values from 23 s to 96 s approximately.

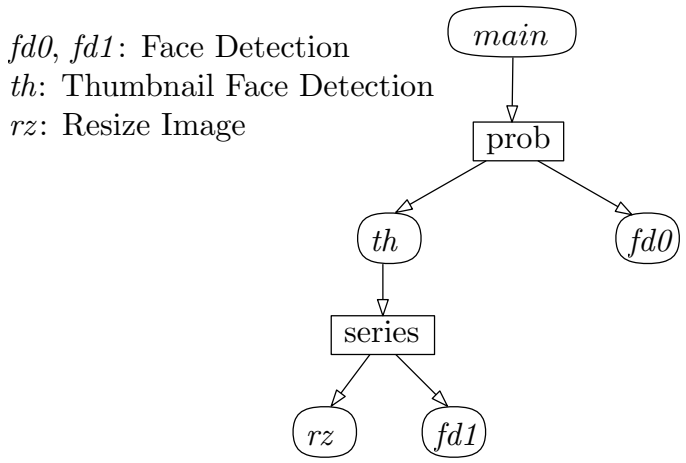


Figure 5.1: Execution Dependency Tree model of the FD application.

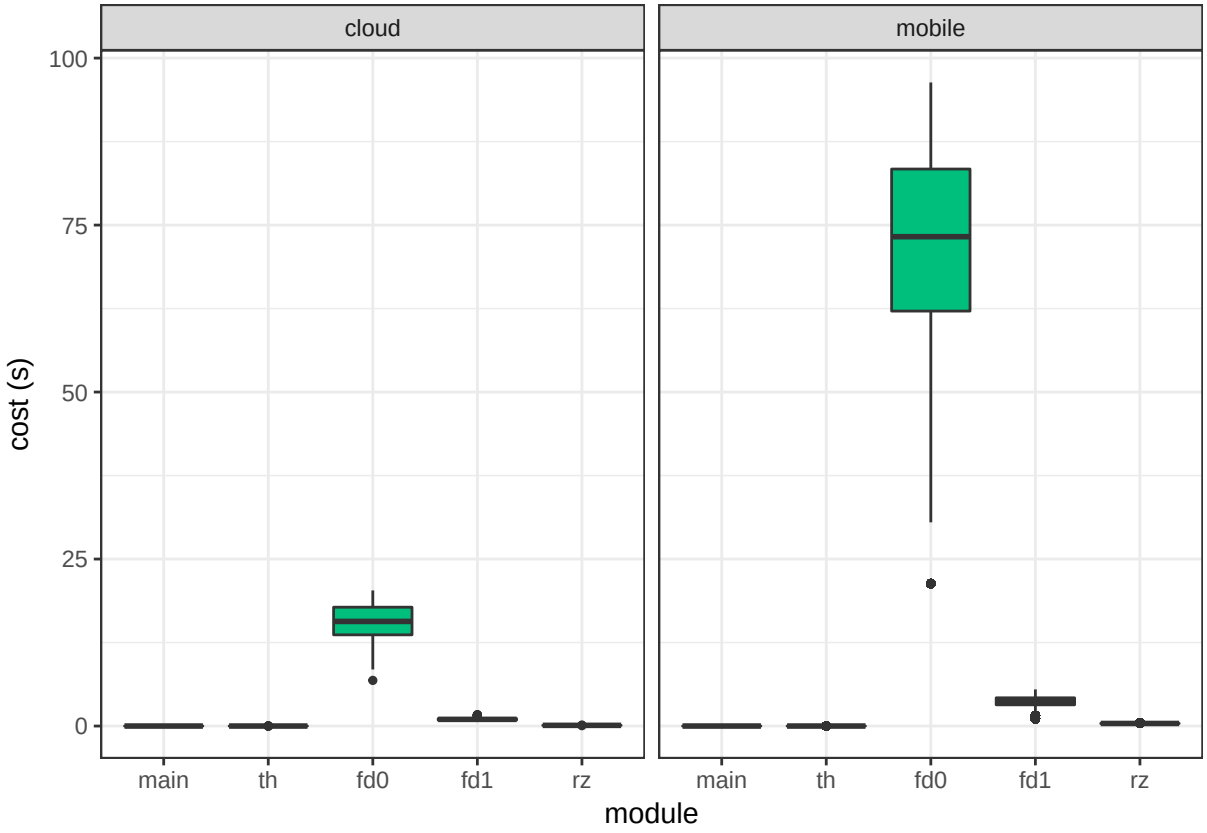


Figure 5.2: Profiled module execution costs for the FD application.

Figure 5.3 shows the boxplots of the transmission costs of each module as profiled from runs of the application for 200 images. The transmission costs of modules main and fd1 are 0 MiB and 0.39 MiB, respectively. The costs for modules th and fd0 have a mean of 4.07 MiB, and for rz, the cost is 4.71 MiB in average. The costs have outliers at roughly 1.25 MiB, 2 MiB, 2.5 MiB, 8.4 MiB and 9.23 MiB.

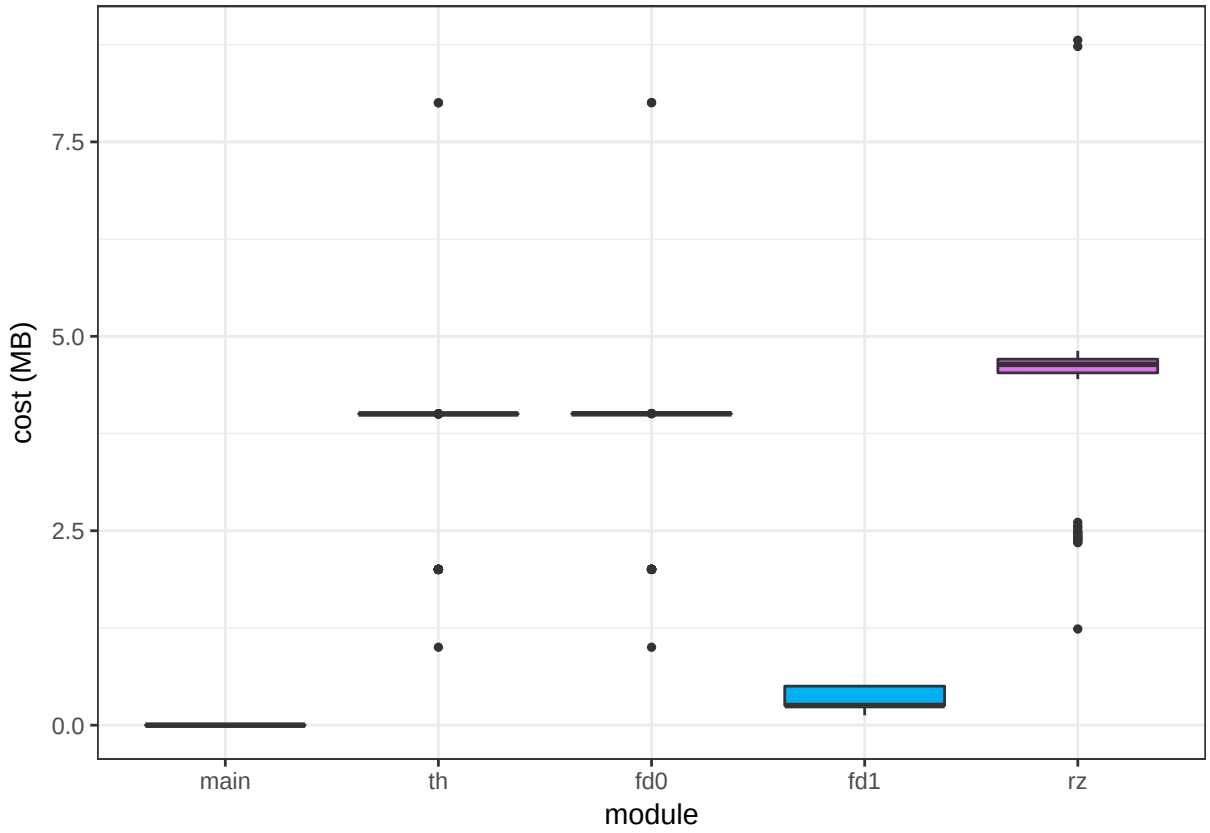


Figure 5.3: Profiled module transmission costs for the FD application.

Table 5.3 shows the parameters of the robust method computed from the interval of costs of the modules. The intervals are computed using the average and maximum profiled costs for the module. The parameters for module fd0, i.e., performing face detection on a full image on the mobile, are $\beta_{\text{fd0}} = 71\,850.95$ ms (the module's average cost), and $\delta_{\beta_{\text{fd0}}} = 96\,350$ ms $- \beta_{\text{fd0}} = 24\,499.05$ ms (the module's maximum cost minus its average cost).

Randomly Generated Application

We evaluate the novel method on large application of 100 module nodes that are connected by series dependencies. We restrict 1 % of the module nodes to run in the mobile device,

Module	α (ms)	δ_α (ms)	β (ms)	δ_β (ms)	σ (MiB)	δ_σ (MiB)
main	0.01	0.01	0.01	0.01	0.00	0.00
th	0.10	7.90	0.05	4.95	4.07	4.33
fd0	15591.07	4709.93	71850.95	24499.05	4.07	4.33
fd1	1017.97	682.03	3657.28	1824.72	0.39	0.14
rz	99.91	48.09	383.27	89.73	4.71	4.52

Table 5.3: Interval-based costs of the modules of the FD application.

and another 1 % to run in the cloud. The cost of the modules are generated randomly using *Gamma Distributions*. This is similar to [55] which uses Exponential Distribution, a particular case of *Gamma*, to model the CPU-time of processes. For example, Figures 5.4 and 5.5 show the boxplots of the execution and transmission costs of 10 modules of the Single-Series application.

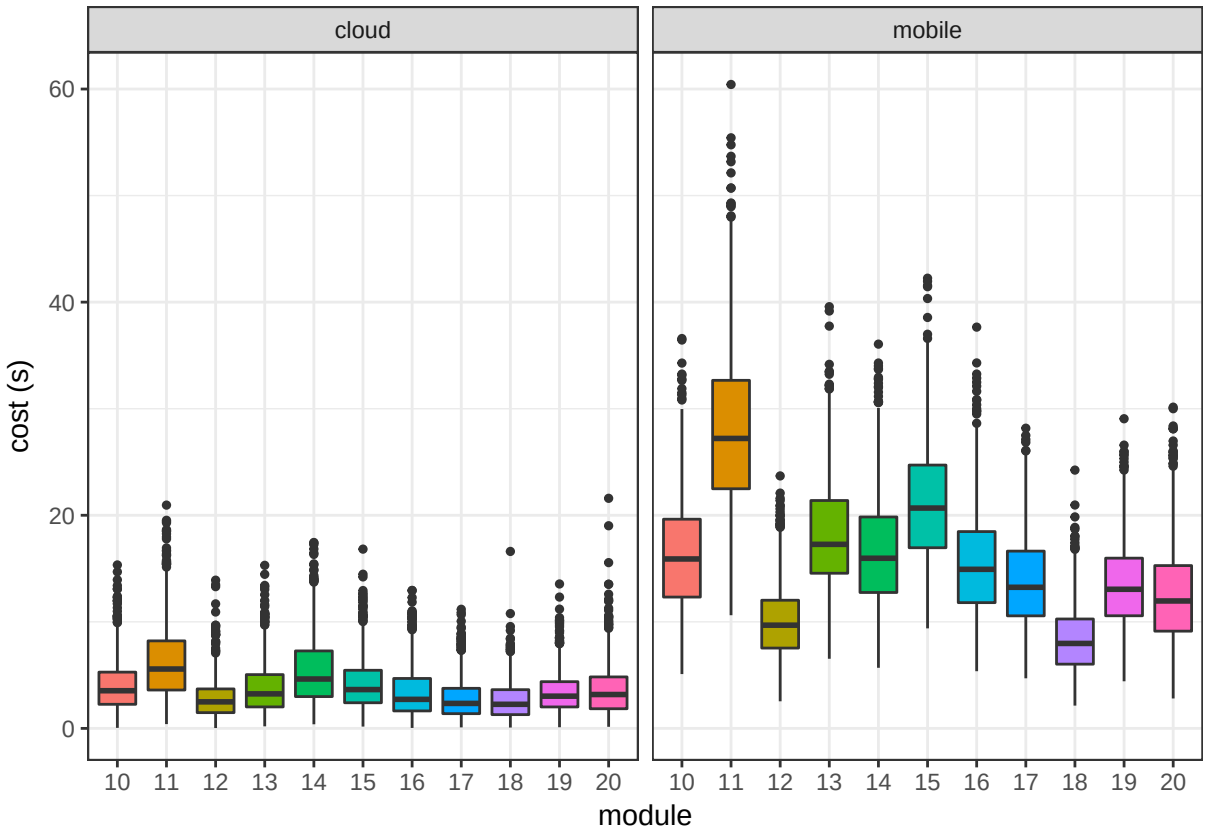


Figure 5.4: Execution costs for some modules of the Single-Series application.

Specifically, the mean execution cost of every module ranges from 2 s to 6 s in the cloud, and from 6 s to 30 s in the mobile. For each module, the average execution cost in the mobile is between three and five times the average execution cost in the cloud. The

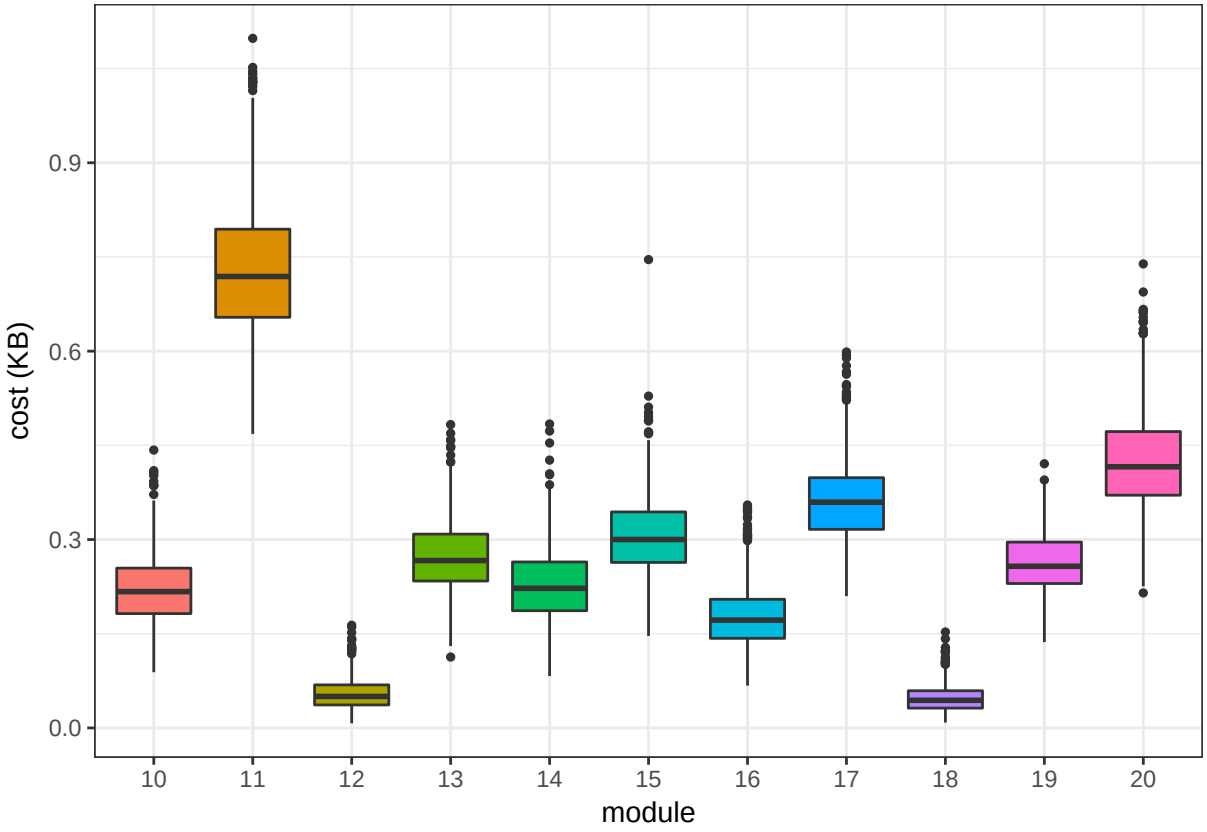


Figure 5.5: Transmission costs for some modules of the Single-Series application.

mean transmission cost for each module is from 2 B to 120 B . The variance of each cost is between 1 and 4 times the average of the cost. All these properties are summarized as follows. First, $\alpha_i \in [2, 6]$, $\beta_i \in [6, 30]$, $3\alpha_i \leq \beta_i \leq 5\alpha_i$. Using the network bandwidth $r = 40$ bit/s, for example, results in a transmission cost from 0.4 s to 24 s, i.e., $\zeta_i \in [0.4, 24]$. Further, $\alpha_i \leq \zeta_i/r \leq 20\alpha_i$. Finally, the variance of each cost is $\sigma^2_{\alpha_i} \in [\alpha_i, 4\alpha_i]$, $\sigma^2_{\beta_i} \in [\beta_i, 4\beta_i]$ and $\sigma^2_{\zeta_i} \in [\zeta_i, 4\zeta_i]$.

5.4.2 Robust vs Average-based cost Optimum

Figure 5.6 shows the execution cost of 1000 runs of the FD application for two different offloading solutions and network bandwidths. Both the non-robust and robust solution are the same for every bandwidth considered other than 0.5 MiB/s. The reason is that the EDT of the FD application has a small number of nodes. For a network data rate of 0.5 MiB/s, the robust and non-robust solutions are different. The non-robust solution is equivalent to the mobile-only solution, and the robust solution offloads module fd0 (both offloading decisions are discussed in Section 5.4.3). We analyze the cost of the application for this data rate in more details next.

Figure 5.7 shows the CDFs of the application running costs for both methods at a network speed of 0.5 MiB/s. For both methods, one half of the executions finished before 15 s which corresponds to when the user performs face detection on a resized version of the image. The other half takes more time, typically from 40 s to 95 s. However, we observe that the robust solution for $\Gamma = 9$ has the effect of lowering the expected worst cost for the application. For instance, all but 7 executions (i.e., 99.3 %) finished before 87 s approximately, which is faster than the 99.3 % for the solution with $\Gamma = 0$ (about 98 s). Minimizing the expected worst case instead of the average cost is crucial for some scenarios such as concerning the user perception of the application performance. For instance, the experiments in [28] showed that more than half of the users in the study considered the real time game application frustrating or unusable when the execution time was higher than 373 ms.

The reason why the 7 runs of the application that take the highest cost (about 150 s) for $\Gamma = 9$, is that the robust formulation minimizes the *average* or *expected* worst case application cost. Here, *average* results from considering the probabilistic dependencies when computing the application cost estimate, which uses the weighted sum of the module's cost, in particular, the probability factors ρ_i in Eq. (5.7). However, we must emphasize that the goal of our robust formulation is to minimize the expected execution cost while

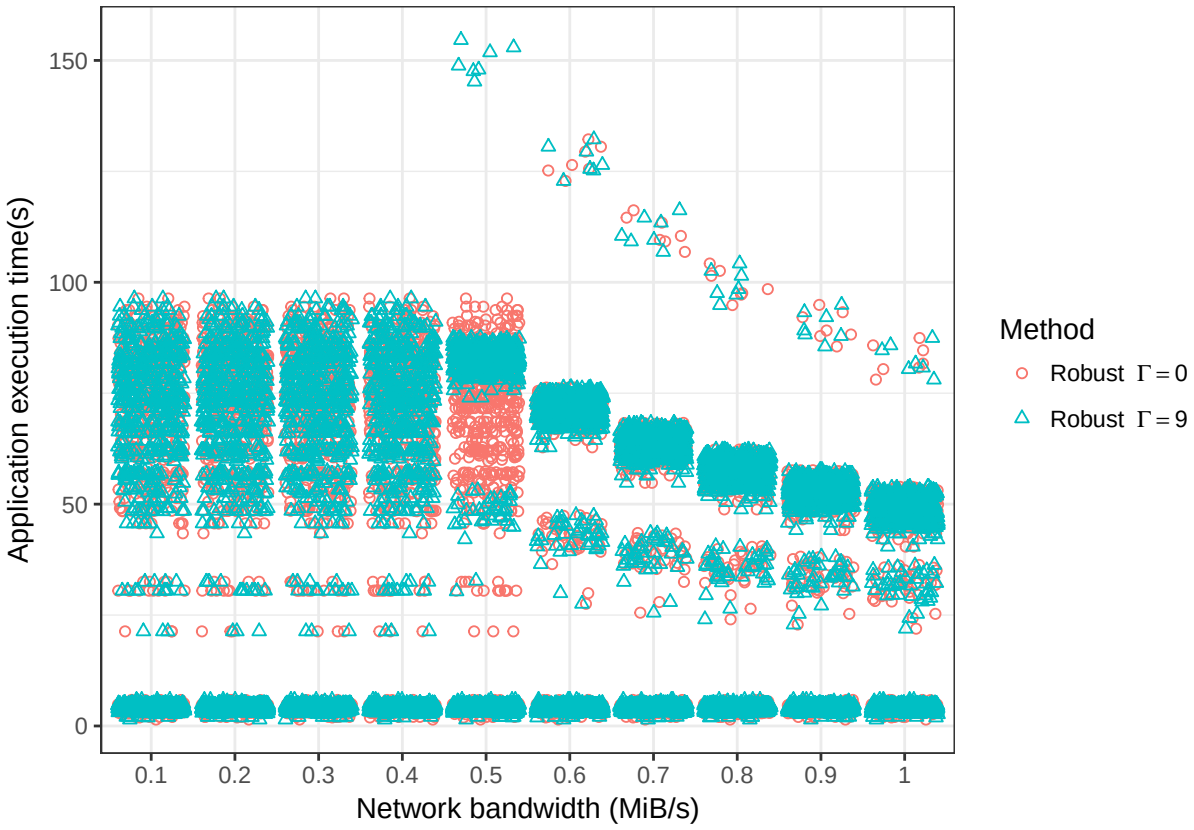


Figure 5.6: Execution cost of the FD application.

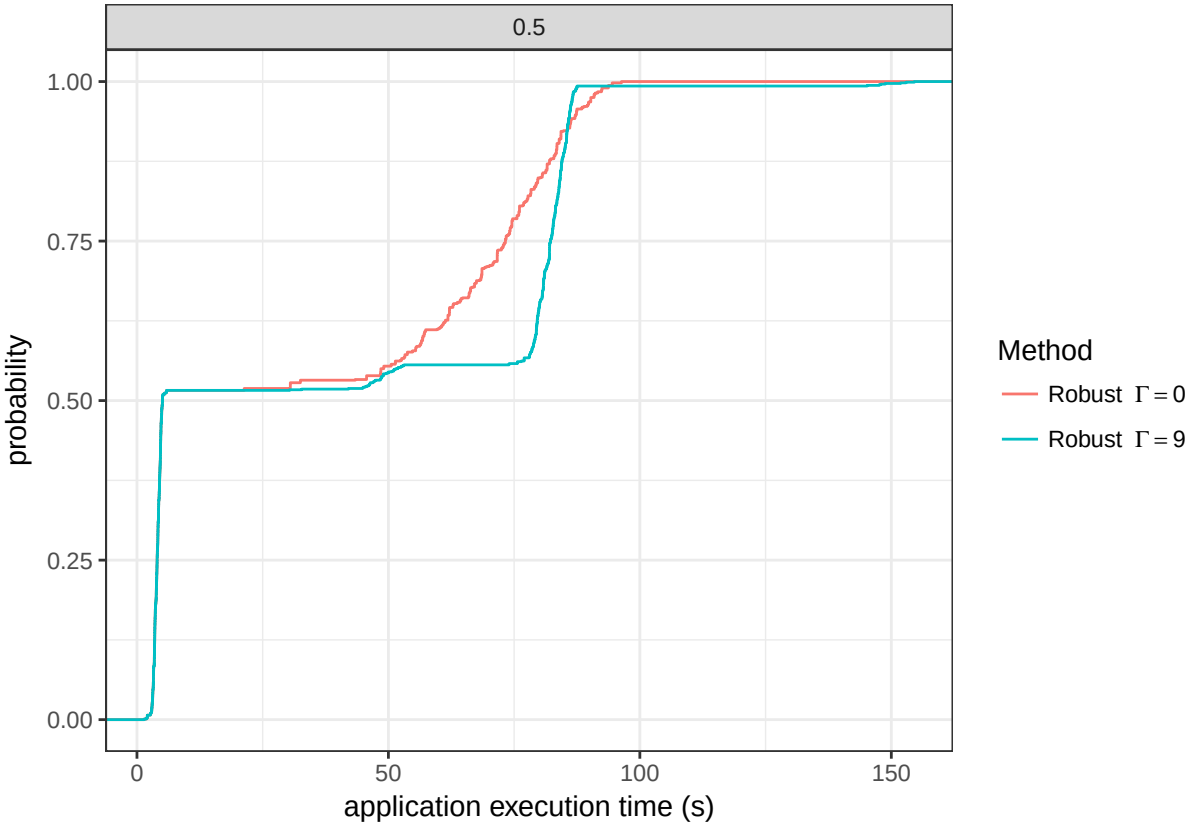


Figure 5.7: CDF of the costs for the FD application for a network speed of 0.5 MiB/s.

being robust to changes in the parameters. This formulation is different than minimizing the worst case cost[1] of the application.

5.4.3 Effect of robustness coefficient Γ

Figure 5.8 shows the robust solutions for the FD application for different degrees of robustness Γ for different network bandwidths. For the bandwidths slower than 0.5 MiB/s, the non-robust and all the robust solutions are the mobile-only decision. Similarly, for bandwidths of 0.6 MiB/s and more, the non-robust and all the robust solutions are the same, corresponding to running fd0 on the cloud, and all the other modules in the mobile. The robustness coefficient does not make a big impact on this scenario as the application's EDT has a small number of nodes.

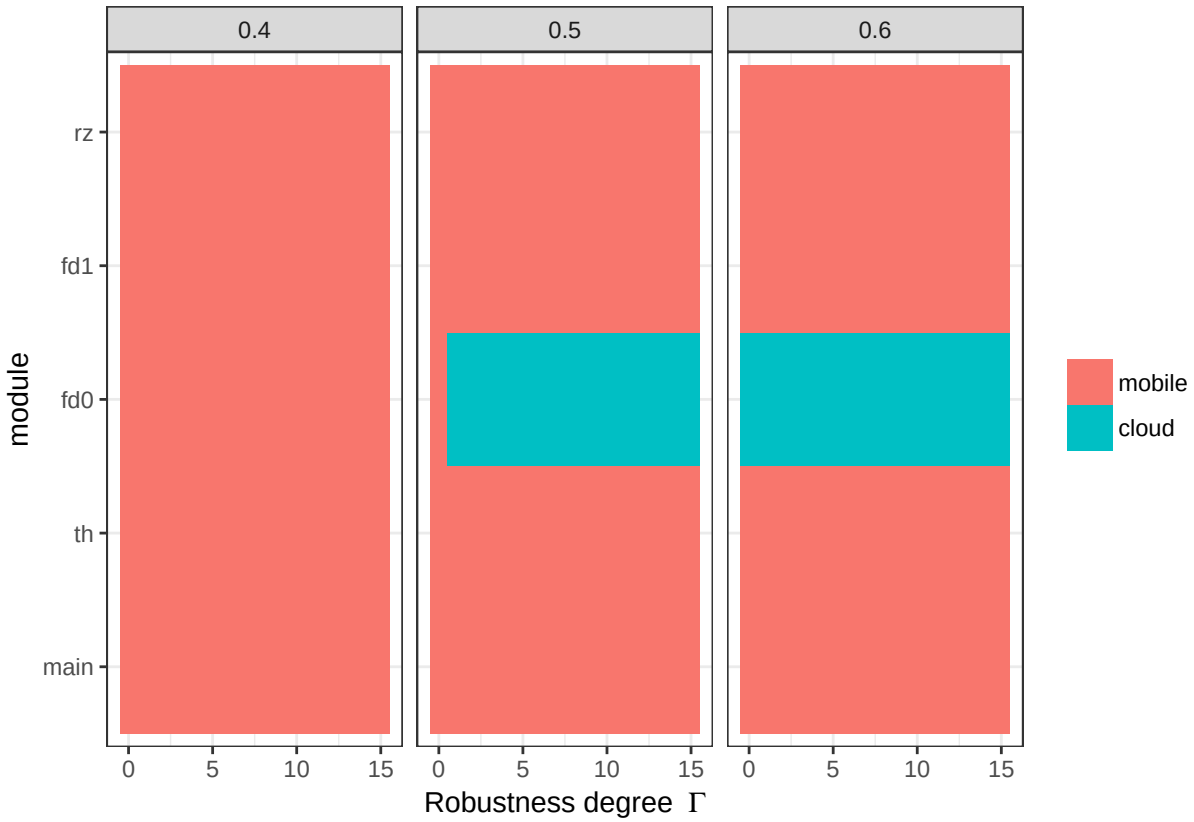


Figure 5.8: Robust solutions for the FD application for three network bandwidths.

The impact of the coefficient Γ, however, is higher for robust solutions for the Single-

[1]The problem of minimizing the worst case cost can be easily achieved by using the EDT model and a cost model similar to the *average-based* cost model, that uses the maximum cost (instead of the average) of each module and specifies appropriate functions for the cost of the dependency nodes.

Series application. Figure 5.9 shows the offloading decisions for the Single-Series application for different Γ grouped by network bandwidth. We see that increasing Γ for the same bandwidth results in changing the offloading decision for some modules from mobile to cloud and vice versa. The change for each module depends on the level of uncertainty of each cost of the module. We also notice that increasing the network bandwidth generally results in robust solutions that offload a higher number of modules.

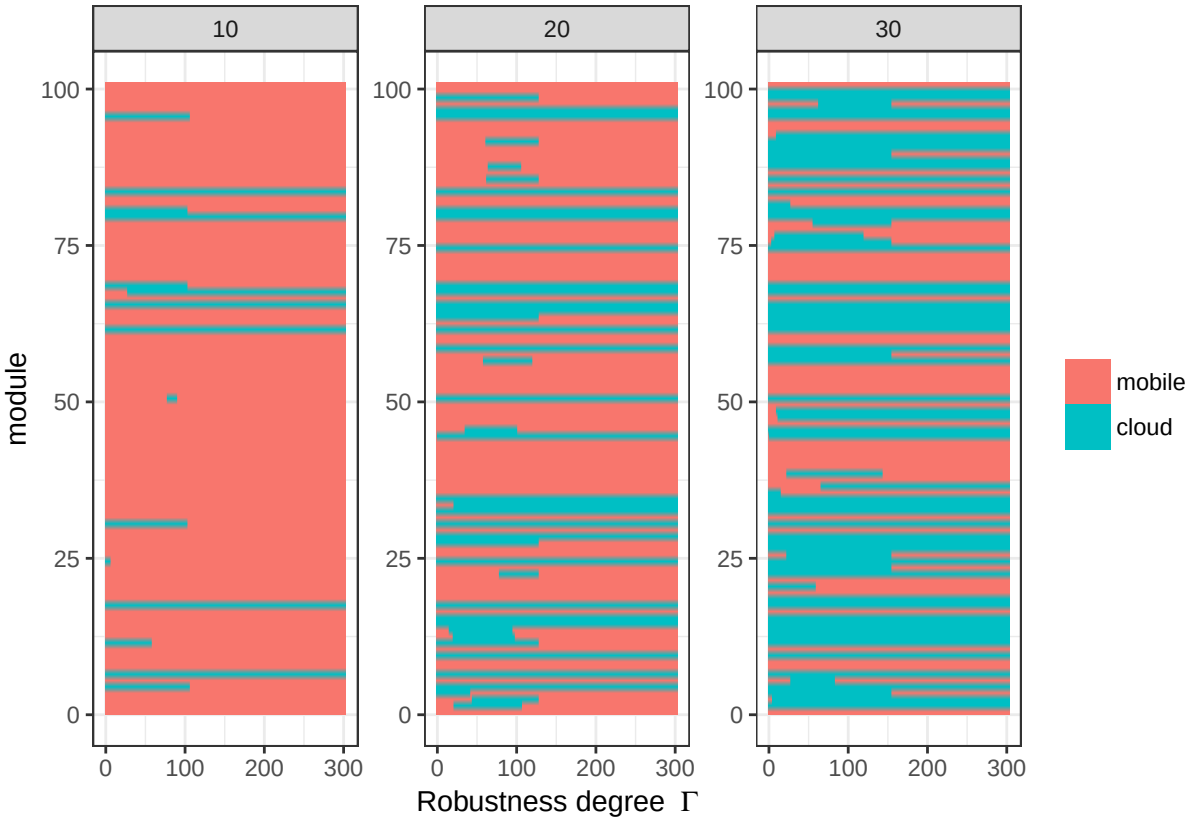


Figure 5.9: Robust solutions for different Γ for the Single-Series application and three network bandwidths.

Figure 5.10 shows the number of offloaded modules per Γ and network data rate for the Single-Series application. We can distinguish three main trends in general. Let us consider data rate of 20 bit/s. First, from $\Gamma = 1$ to 80, the number of modules that are offloaded slowly increases from 29 to 40. Second, the inverse trend is seen from $\Gamma = 80$ to 125, i.e., increasing Γ slightly decreases the number of modules that are offloaded from 40 to 34. Finally, at $\Gamma = 125$, the number of offloaded modules suddenly decreases to 26, and the all robust solutions are the same for $\Gamma > 125$. These trends are more or less apparent depending on the network bandwidth. This changes are related to the tradeoff in each node between (1) the difference in execution costs for mobile and cloud locations, and (2)

the transmission cost of the module.

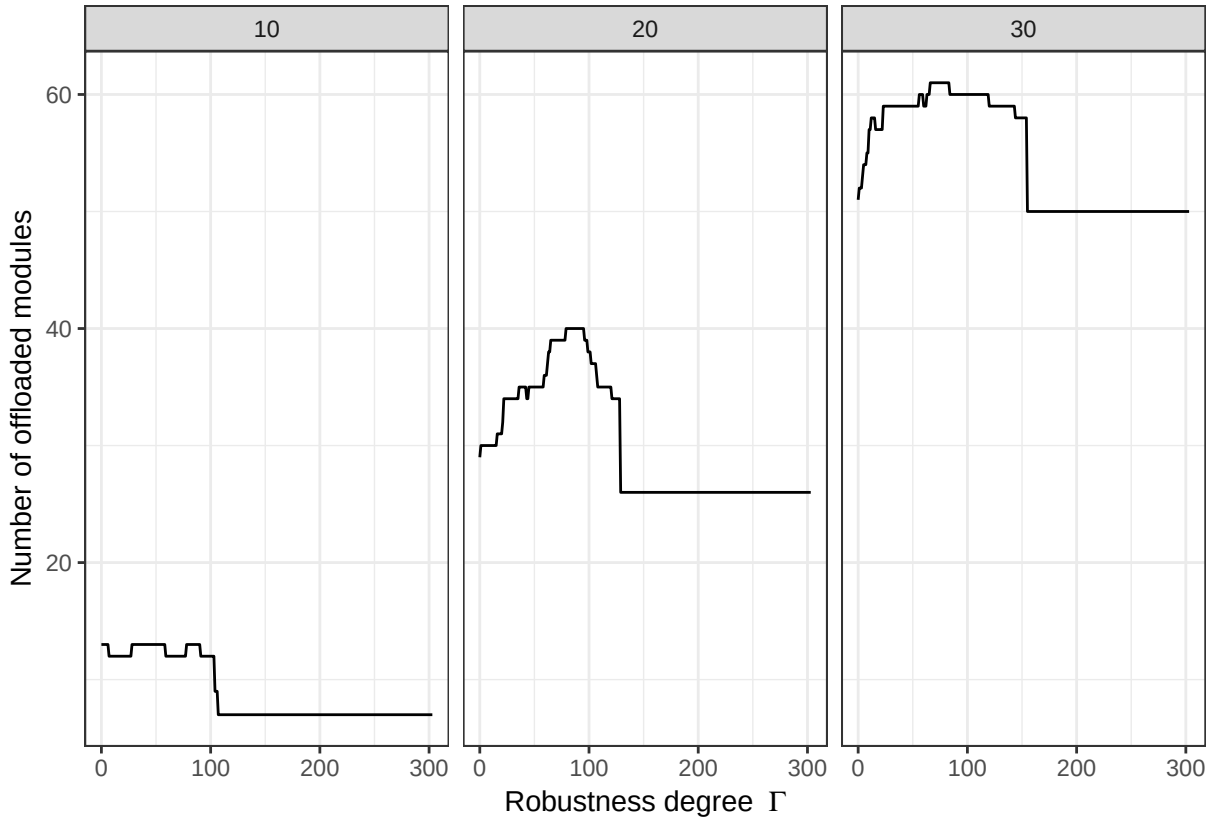


Figure 5.10: Number of offloaded modules per robust coefficient and three network bandwidths for the Single-Series application.

Figure 5.11 shows the distribution of application costs for 1000 executions of the Single-Series application for the robust solution for each robustness coefficient Γ. Each subfigure corresponds to a different network bandwidth. Each figure represents, with a color gradient, the number of executions per coefficient Γ and time. The lighter color represents a higher number of application executions. For example, for the data rate of 10 bit/s and $\Gamma \geq 100$, the execution of the application often takes 1.47 s (the lighter color). We can see that in general, all the robust solutions show a distribution of application costs that is similar to the non-robust optimal solution (i.e., $\Gamma = 0$). We can also see that lower values of Γ result in lower application cost. Next, we take a closer look at these costs distributions.

Figure 5.12 depicts the density function of the costs of the application for the robust solutions for a network bandwidth of 60 bit/s. It can be seen that the execution cost for the application is lower for the robust solution with $\Gamma = 9$ even compared to the non-robust solution, while it is higher for $\Gamma = 300$. This example illustrates the importance of being able to relax the level of conservatism of the robust solution.

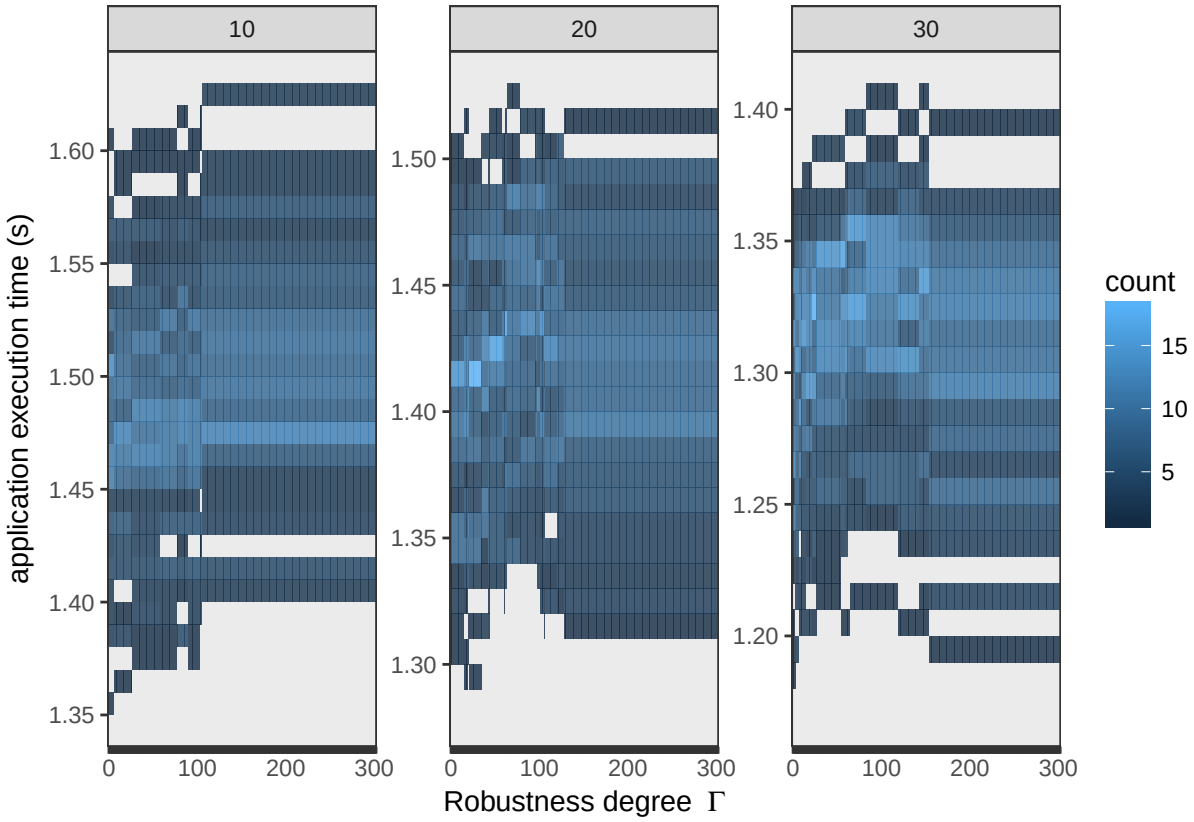


Figure 5.11: Distribution of costs for 1000 executions of the Single-Series application for each robust solution of robustness degree Γ and different network bandwidths.

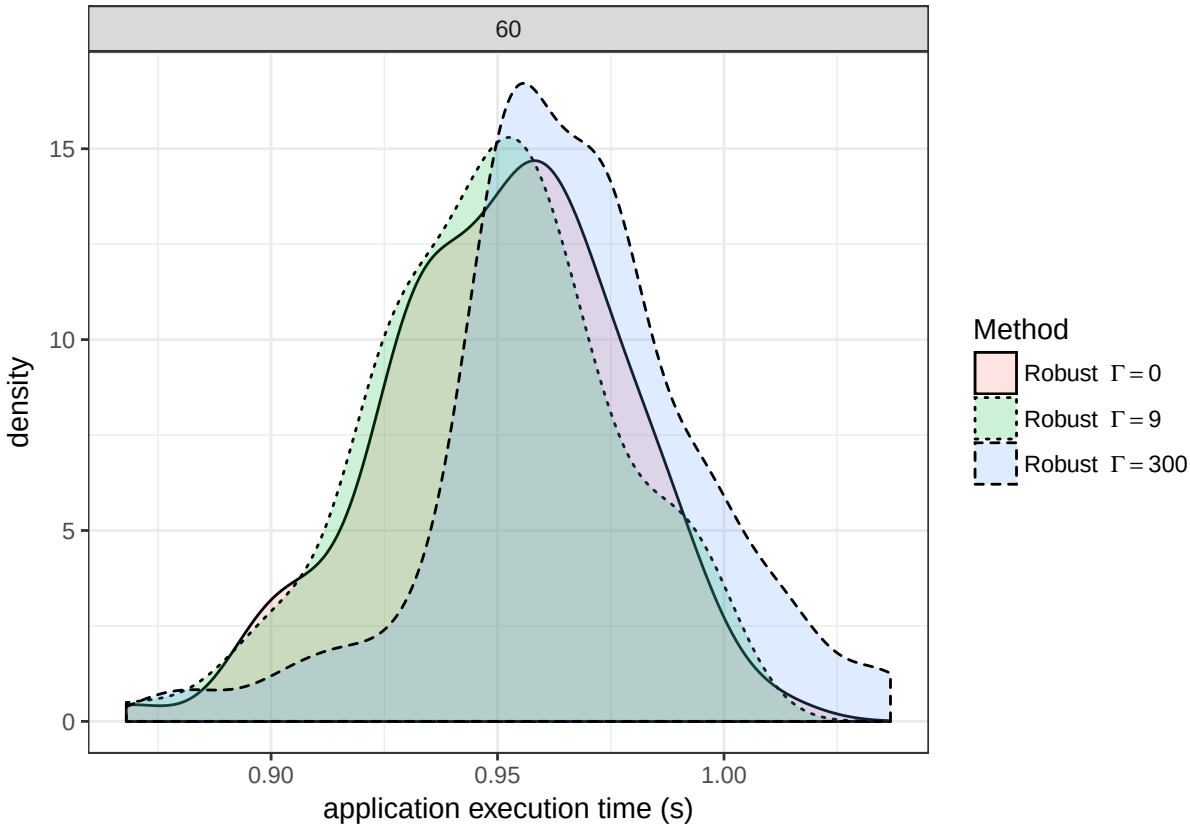


Figure 5.12: Density function of costs for 1000 executions of the Single-Series application for robust solutions of robustness degree Γ and network bandwidth of 60 bit/s.

5.4.4 Tractability of the problem

We implemented the robust Algorithm 7 in Matlab/Octave. The LP problem in Lines 11 and 12 is solved via the *glpk* function with a limit of 100 Simplex iterations. The simulations are run with Octave on a Intel *iCore 5* CPU with 4 cores.

Figure 5.13 shows the execution time of the algorithm for the Single-Series application, which has 100 nodes, for different network bandwidths. We see that the algorithm execution time is lineal with respect to the robust coefficient Γ.

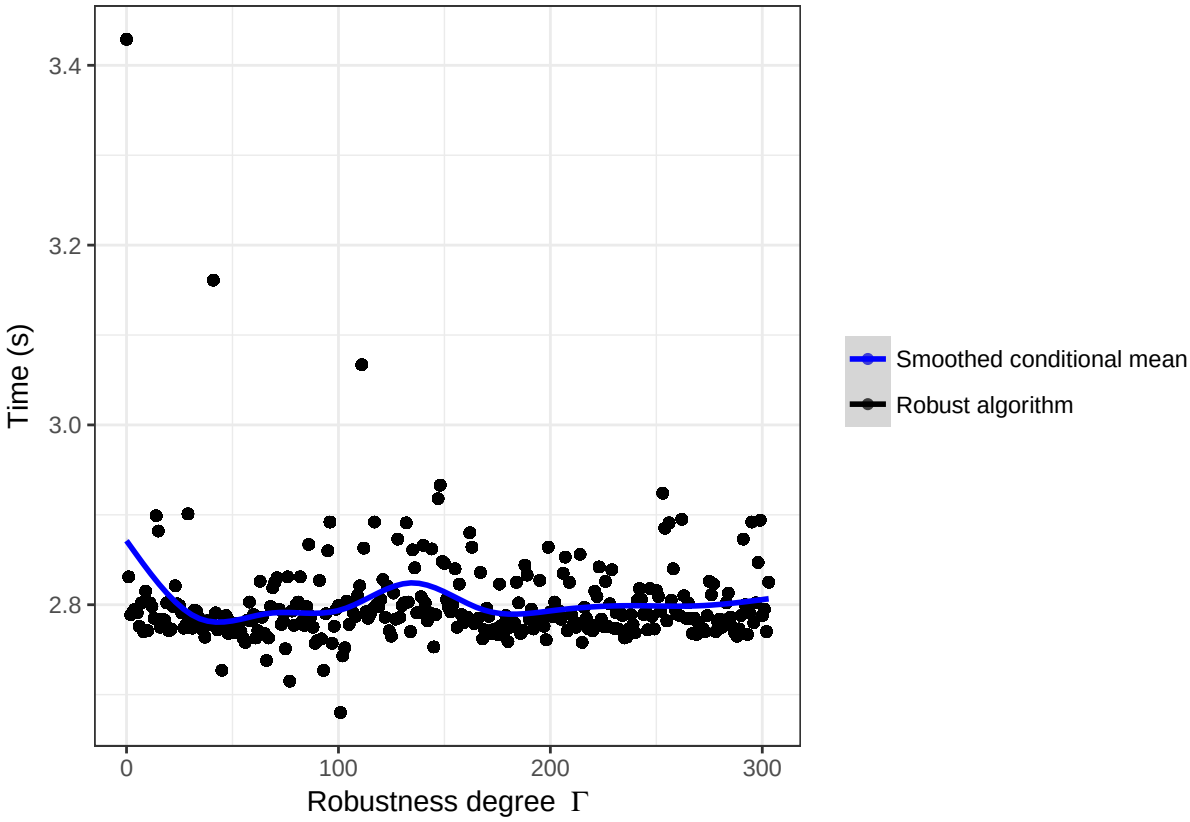


Figure 5.13: Execution time of the robust algorithm for the Single-Series application.

Figure 5.14 shows the execution time of the algorithm for different application sizes. The algorithm running time was approximated by a polynomial of degree 2 with an approximation error of $r^2 = 0.994$, and of degree 3 with an approximation error of $r^2 = 0.998$.

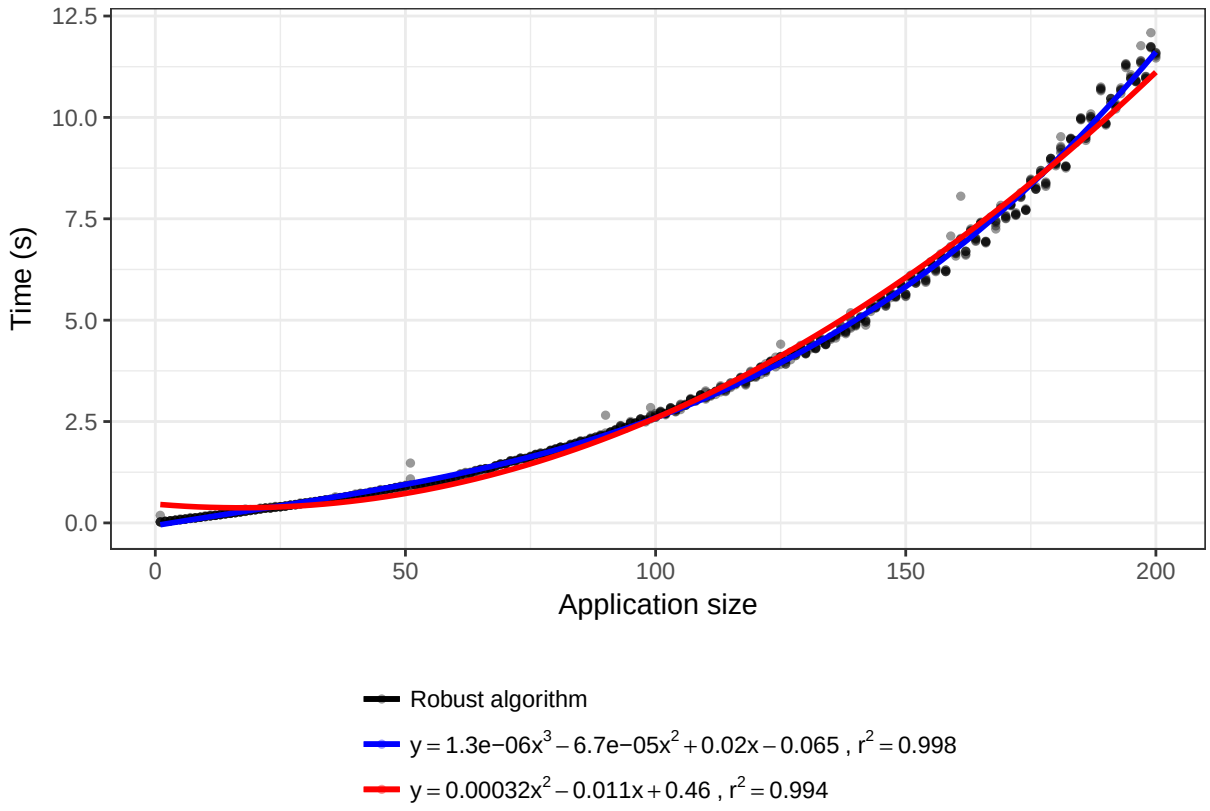


Figure 5.14: Execution time of the robust algorithm vs application size.

5.5 Summary

We have presented a novel formulation of the application offloading problem that addresses uncertainty via robust optimization. The application is modeled via the EDT application model, and each module costs take values within cost intervals that are computed considering the uncertainty. The robust formulation aims at finding an offloading solution that minimizes the expected application cost when a given number Γ of costs may take the worst case value within its interval. The robust method for $\Gamma = 0$ is equivalent to the non-robust problem where the costs of the modules are the lower bound of the cost intervals. However, the robust formulation additionally supports global constraints, in contrast to the dynamic programming method proposed in Section 3.4 that also uses EDT and average-based cost model. The robust algorithm solves a polynomial number of LP problems. In our simulations, the robust offloading solution achieved similar application cost and the variance than the non-robust formulation for a face detection application and a randomly generated application with 100 modules. Further, it showed an execution cost that was closely approximated by a second degree polynomial with respect to the number of application modules. As future work, we should investigate in which scenarios with uncertainty the proposed model should be used instead of a non-robust approach, for example, for highly variable network bandwidths.

Chapter 6

Conclusions and Future Work

We have addressed the problem of application offloading for Mobile Cloud Computing. Our main goal is to increase the accuracy of the offloading solution. Our main contributions consist of a new application model, three cost models based on the new application model, and three decision algorithms for each cost model. This chapter summarizes these contributions and outlines directions for future work.

6.1 Contributions

We proposed a novel fine grained application model, called Execution Dependency Tree (EDT), that represents the application as a tree graph. Each module in the application is represented by multiple nodes in the tree according to the execution paths that result on the execution of the module. Additionally, the graph models three different types of dependencies between the application modules, specifically, *series*, *probabilistic* and *parallel* module dependencies, via dependency nodes. The main benefits of this model over the state of the art are that (1) it supports multiple offloading decisions per application module based on the execution paths, (2) it supports fine grained cost models where the cost of each module is modelled based on the execution paths, and (3) it supports complex cost

models where the cost of the application is computed considering the three dependencies between modules. Based on the novel application model, we developed three cost models that address different scenarios.

We developed an average-based cost model where the execution and transmission costs of each module are represented by a their average cost values. The cost of running the entire application is then estimated as an average considering the costs and the dependencies between the modules. This model is appropriate for (1) fast offloading decision algorithms, (2) when the costs of modules are relatively stable and (3) when the offloading problem requires minimizing the average cost of the application.

We developed a statistics-based cost model where costs are modelled by random variables and their CDFs. The cost of running the entire application is estimated by a CDF that is computed using the CDF of the costs of the modules and according to the dependencies between the modules as expressed by the EDT. The model captures the random nature of the execution costs of the application and supports optimization functions and constraints based on statistical measurements, e.g., the expected cost or the 90 percentile. Furthermore, the new model is, to the best of our knowledge, the first model for application offloading to support statistical analysis of the performance of the application. We illustrated the use of the model to analyze the cost of the application under different conditions such as network data rates, device speed up and user behavior. We verified the accuracy of the model for different dependency types, number of nodes and random variables with different variance. In our evaluations, the mean absolute percentage error between average measured and estimated costs was less than 4 % for series dependency and probabilistic dependency nodes, where the error was up to 7.23 % for parallel dependencies.

We proposed an interval-based cost model to address uncertainty in the parameters efficiently. The cost of the modules is represented by a cost interval while the cost of the application is estimated as a single value that represents the expected maximum cost. The main benefits of the model are (1) handling uncertainty in a tractable way, (2) providing guarantees regarding a expected maximum application execution cost, and (3) requiring less information regarding the module costs compared to the statistics-based cost model.

The three cost models proposed, i.e., average-based, interval-based, and statistics-based, differ in their assumptions, complexity and the optimization goals that they support. Table 6.1 shows a comparison of these main features. The features include the simplification when modeling the cost, the computational complexity of using the model, and the optimization goals supported by the cost model. In general, the strongest the simplification assumed on the cost, the lower the complexity of the model, and the more limited the

optimization functions supported. For instance, the average-based model has the lowest complexity as costs are represented by the expected value. However, this simplification restricts the optimization goal and constraints to be based on the average application costs. The statistics-based model is very descriptive as costs are modelled by CDFs, including the cost estimated for the entire application. As a result, the model can be used to solve optimization problems based on any statistical function on the CDF of the application cost. This flexibility and accuracy comes at the price, however, of performing operations that are computationally expensive, e.g., the convolution of empirical distribution functions. The interval-based cost model has benefits from the other two cost models as it uses intervals to model the module costs, with medium complexity. What cost model and algorithm we choose depends on our optimization goal, the restrictions, and the resources available to solve the problem.

Cost Model	Cost Simplification	Complexity	Supported Optimization Goals
Average-based	Single value	Low	Average application costs
Interval-based	Intervals	Medium	Mean and worst case costs
Statistics-based	CDFs	High	Statistical functions on the costs

Table 6.1: Comparison of the proposed cost models.

Using the proposed application and average-based cost models, we presented an efficient algorithm that finds the offloading solution that minimizes the application execution cost. The algorithm follows the dynamic programming technique and finds the offloading decision with minimum average application cost. The new approach found offloading decisions that achieved up to 42 % faster average application execution compared to the traditional Call Graph Model (CGM) for a face detection mobile application under different network conditions.

We developed an efficient algorithm for the novel statistics-based cost model for the special case when the optimization function is based on a statistical measurement that meets specific properties. The algorithm follows a dynamic programming approach.

For the interval-based cost model, we presented a tractable formulation of the application offloading problem based on robust optimization. The problem formulation aims at finding an offloading solution with a minimum guaranteed cost for all the possible realizations of a number of costs within their respective uncertainty sets. We proposed an algorithm that solves the robust problem solving a polynomial number of Binary Linear Programming (BLP) optimizations. The novel robust formulation can be applied to scenarios where we need guarantees on the cost of the application while requiring an efficient

algorithm. In our evaluations, the robust solution achieved average application costs similar to those of the optimal solution while achieving lower expected maximum application cost.

The proposed solutions are not restricted to any specific mobile application type or scenario. Example of applications include virtual reality, video games, chess games, face and speech recognition, augmented reality, video rendering, etc. If very fast application response is required such as in video games, the fast average-based cost model is more appropriate. Likewise, the more flexible statistics-based cost model is more suitable for scenarios with fixed deadlines such as chess games. Finally, for highly uncertain scenarios, such as when the mobile user travels on a car or train, and experiences handovers that result in rapid changes in network bandwidth, the robust approach using the interval-based cost model finds offloading solutions that are near optimal for all the values in the uncertainty set.

6.2 Limitations

While the proposed contributions improve the accuracy of the offloading method, they come with some limitations.

Capturing the execution paths of the application in the EDT model may lead to what is known as path explosion problem. This problem, which may affect applications with recursive calls, can be addressed by summarizing recursive execution paths by additional nodes in the EDT tree. For applications with large numbers of modules, the problem may be addressed reducing the number of offloadable modules, for example having the developer specifying the modules candidate for offloading.

The accuracy of the cost models decrease for highly parallel applications as the shared resources, such as the CPU cores in the mobile device, are insufficient to serve all the parallel modules. Other shared resources include the memory and the network interface.

The statistics-based cost model has two main limitations. First, obtaining accurate statistical information of the costs requires either extensive profiling of the application, mobile devices and network, or accurately estimating the costs by known distribution functions. This might not be practical in some scenarios as the costs imposed by the offloading framework to gather this information may negate the benefits of the offloading operations, or there might be no known distribution function that describe the cost of the module. Second, the decision algorithm based on the statistics-based cost model has a high

computational complexity in general as it involves computing convolutions of cumulative distribution functions. The offloading decision algorithm can be run in the server and the decision communicated to the mobile device.

A limitation of the robust formulation is that it supports the series and probabilistic dependencies of the EDT application model but not the parallel dependency. Clearly, this limitation only affects parallel applications running on multi-core mobile devices. These scenarios can be addressed updating the optimization function. This change, however, implies an redesign of the offloading decision algorithm, and even more, it might result in an intractable robust formulation. Another approach is to model parallel executions as sequential, which negatively impacts the accuracy of the model.

6.3 Future Work

As a future work we envision the following main directions.

We want to improve the application model for complex application execution paths, particularly for recursive calls, which can result in long execution paths and thus extremely long branches in the EDT tree. To shorten these branches, a set of recursive calls can be summarized in a special node in the tree. For example, having an execution path of $n > 0$ recursive calls, we can summarize the last $m \leq n$ calls as one tree node. This new node summarizes the set of recursive calls and their respective costs. An interesting question here is what the value m should be. The decision should take into account the tradeoff between the computational and the transmission cost of the m calls. An advantage of this approach is that it requires no changes in the cost models.

We also want to increase the accuracy of the average-based cost model for highly parallel applications. These applications may be more common in the near future due to new paradigms for application development for cloud computing such as micro services. This can be achieved by including information of the environment in the cost model, such as number of cores in the mobile device and the cloud. Another approach is to address the problem of offloading to multiple locations where each location represents a single CPU core of either the mobile device or the cloud server.

Concerning the uncertainty, we want to investigate the effect of including information of the recent past in order to estimate the uncertainty sets. This may be appropriate, for example, when the mobile user is connected a public access point where several other mobile users continuously connect and disconnect. As a consequence, the network data

rate perceived by the user fluctuates with the number of users connected to the access point.

In summary, we have proposed novel solutions for the application offloading problem. While these results contribute to more accurate application offloading, they also raise new research questions.

Bibliography

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

[2] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[3] Cong Shi, Mostafa H. Ammar, Ellen W. Zegura, and Mayur Naik. Computing in cirrus clouds: The challenge of intermittent connectivity. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 23–28, New York, NY, USA, 2012. ACM.

[4] Antonios Litke, Dimitrios Skoutas, and Theodora Varvarigou. Mobile grid computing: Changes and challenges of resource management in a mobile grid environment. In *5th International Conference on Practical Aspects of Knowledge Management (PAKM 2004)*, 2004.

[5] Cisco VNI Mobile. Cisco visual networking index: Global mobile data traffic forecast update, 2013–2018. *San Jose, CA*, 2014.

[6] Cisco VNI Mobile. Cisco visual networking index: Global mobile data traffic forecast update, 2015–2020. *San Jose, CA*, 2016.

[7] Google. Google drive [online]. Available: https://www.google.com/drive/, Accessed: January 2017.

[8] Amazon. Amazon elastic compute cloud (amazon EC2) [online]. Available: https://aws.amazon.com/ec2/, Accessed: January 2017.

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

[10] Shumao Ou, Kun Yang, and Antonio Liotta. An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In *Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06)*, pages 10 pp.–125, March 2006.

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

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

[13] B. K. Huang, C. C. Cheng, C. H. Lin, and P. C. Hsiu. A cloud-based offloading service for computation-intensive mobile applications. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 80–89, Aug 2015.

[14] J. Barrameda and N. Samaan. A novel application model and an offloading mechanism for efficient mobile computing. In *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 419–426, Oct 2014.

[15] J. Barrameda and N. Samaan. A novel statistical cost model and an algorithm for efficient application offloading to clouds. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2017.

[16] Jose Barrameda and Nancy Samaan. A robust formulation for efficient application offloading to clouds. *IEEE Transactions on Cloud Computing*, 2017, Submitted.

[17] Dropbox. Dropbox [online]. Available: http://www.dropbox.com/, Accessed: January 2017.

[18] Apple. Siri [online]. Available: http://www.apple.com/ca/ios/siri/, Accessed: January 2017.

[19] Google. Google assistant [online]. Available: https://assistant.google.com/, Accessed: January 2017.

[20] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[21] C. Mei, D. Taylor, C. Wang, A. Chandra, and J. Weissman. Sharing-aware cloud-based mobile outsourcing. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 408–415, June 2012.

[22] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[23] Google. Gmail [online]. Available: http://www.gmail.com/, Accessed: January 2017.

[24] Gonzalo Huerta-Canepa and Dongman Lee. 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, 2010. ACM.

[25] Daniel C Doolan, Sabin Tabirca, and Laurence T Yang. Mmpi a message passing interface for the mobile environment. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '08, pages 317–321, New York, NY, USA, 2008. ACM.

[26] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, pages 145–154, New York, NY, USA, 2012. ACM.

[27] Heba Kurdi. Personal mobile grids: Ubiquitous grid environments for personal users. *Journal of Ubiquitous Systems & Pervasive Networks*, 6(1):01–10, 2015.

[28] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by

mobile users. In *2012 IEEE International Conference on Pervasive Computing and Communications*, pages 122–127, March 2012.

[29] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing – a key technology towards 5G. *ETSI White Paper*, 11(11):1–16, 2015.

[30] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, October 2016.

[31] Apache Software Foundation. Hadoop [online]. Available: https://hadoop.apache.org/, Accessed: January 2017.

[32] S. Abolfazli, Z. Sanaei, M. Alizadeh, A. Gani, and F. Xia. An experimental analysis on cloud-based mobile augmentation in mobile cloud computing. *IEEE Transactions on Consumer Electronics*, 60(1):146–154, February 2014.

[33] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *IEEE Communications Surveys Tutorials*, 16(1):369–392, First 2014.

[34] Muhammad Shiraz, Saeid Abolfazli, Zohreh Sanaei, and Abdullah Gani. A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, 63(3):946–964, Mar 2013.

[35] R. Buyya, R. N. Calheiros, J. Son, A. V. Dastjerdi, and Y. Yoon. Software-defined cloud computing: Architectural elements and open challenges. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1–12, Sept 2014.

[36] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 85–90, New York, NY, USA, 2009. ACM.

[37] T. Truong-Huu and C. K. Tham. A novel model for competition and cooperation among cloud providers. *IEEE Transactions on Cloud Computing*, 2(3):251–265, July 2014.

[38] L. Yu, W. T. Tsai, X. Wei, J. Gao, T. T. Hildebrandt, and X. Q. Guo. Modeling and analysis of mobile cloud computing based on bigraph theory. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 67–76, April 2014.

[39] G. Orsini, D. Bade, and W. Lamersdorf. Computing at the mobile edge: Designing elastic android applications for computation offloading. In *2015 8th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 112–119, Oct 2015.

[40] Chathura M. Sarathchandra Magurawalage, Kun Yang, Ritosa Patrik, Michael Georgiades, and Kezhi Wang. A resource management protocol for mobile cloud using auto-scaling. *CoRR*, abs/1701.00384, 2017.

[41] A. N. Al-Shuwaili, A. Bagheri, and O. Simeone. Joint uplink/downlink and offloading optimization for mobile cloud computing with limited backhaul. In *2016 Annual Conference on Information Science and Systems (CISS)*, pages 424–429, March 2016.

[42] S. Yu, R. Langar, W. Li, and X. Chen. Coalition-based energy efficient offloading strategy for immersive collaborative applications in femto-cloud. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.

[43] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In Jean M. Bacon and Brian F. Cooper, editors, *Middleware 2009: ACM/IFIP/USENIX, 10th International Middleware Conference, Urbana, IL, USA, November 30 – December 4, 2009. Proceedings*, pages 83–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[44] Xinwen Zhang, Sangoh Jeong, Anugeetha Kunjithapatham, and Simon Gibbs. Towards an elastic application model for augmenting computing capabilities of mobile platforms. In Ying Cai, Thomas Magedanz, Minglu Li, Jinchun Xia, and Carlo Giannelli, editors, *Mobile Wireless Middleware, Operating Systems, and Applications: Third International Conference, Mobilware 2010, Chicago, IL, USA, June 30 - July 2, 2010. Revised Selected Papers*, pages 161–174. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[45] H. J. Jeong and S. M. Moon. Offloading of web application computations: A snapshot-based approach. In *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*, pages 90–97, Oct 2015.

[46] Paramvir Bahl, Richard Y. Han, Li Erran Li, and Mahadev Satyanarayanan. Advancing the state of mobile cloud computing. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, pages 21–28, New York, NY, USA, 2012. ACM.

[47] Dejan Kovachev, Yiwei Cao, and Ralf Klamma. Mobile cloud computing: A comparison of application models. *CoRR*, abs/1107.4940, 2011.

[48] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *Proceedings Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 40–46, 2002.

[49] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *2013 Proceedings IEEE INFOCOM*, pages 1285–1293, April 2013.

[50] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[51] J. B. Dennis. A parallel program execution model supporting modular software construction. In *Proceedings. Third Working Conference on Massively Parallel Programming Models (Cat. No.97TB100228)*, pages 50–60, Nov 1997.

[52] O. Munoz, A. Pascual-Iserte, and J. Vidal. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology*, 64(10):4738–4755, Oct 2015.

[53] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri. Run time application repartitioning in dynamic mobile cloud environments. *IEEE Transactions on Cloud Computing*, 4(3):336–348, July 2016.

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

[55] R. A. Sahner and K. S. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, SE-13(10):1105–1114, Oct 1987.

[56] L. Deboosere, P. Simoens, J. De Wachter, B. Vankeirsbilck, F. De Turck, B. Dhoedt, and P. Demeester. Grid design for mobile thin client computing. *Future Generation Computer Systems*, 27(6):681 – 693, 2011.

[57] Citrix Systems Inc. Citrix independent computing architecture (ica) [online]. Available: http://www.citrix.com/, Accessed: January 2017.

[58] Microsoft Corporation. Windows remote desktop protocol (RDP) [online]. http://msdn.microsoft.com/En-US/library/aa383015.aspx, Accessed: January 2017.

[59] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan 1998.

[60] TeamViewer Inc. Teamviewer [online]. Available: http://www.teamviewer.com/, Accessed: January 2017.

[61] J. Flinn, SoYoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 217–226, 2002.

[62] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, MobiSys '03, pages 273–286, New York, NY, USA, 2003. ACM.

[63] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, Jul 2003.

[64] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: Enabling interactive perception applications on mobile devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 43–56, New York, NY, USA, 2011. ACM.

[65] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *SIGMETRICS Perform. Eval. Rev.*, 40(4):23–32, April 2013.

[66] Ejaz Ahmed, Adnan Akhunzada, Md Whaiduzzaman, Abdullah Gani, Siti Hafizah Ab Hamid, and Rajkumar Buyya. Network-centric performance analysis of runtime application migration in mobile cloud computing. *Simulation Modelling*

Practice and Theory, 50:42 – 56, 2015. Special Issue on Resource Management in Mobile Clouds.

[67] Y. Wen, W. Zhang, and H. Luo. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *2012 Proceedings IEEE INFOCOM*, pages 2716–2720, March 2012.

[68] Asaf Cidon, Tomer M. London, Sachin Katti, Christos Kozyrakis, and Mendel Rosenblum. Mars: Adaptive remote execution for multi-threaded mobile devices. In *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds*, MobiHeld '11, pages 1:1–1:6, New York, NY, USA, 2011. ACM.

[69] H. Liang, D. Huang, L. X. Cai, X. Shen, and D. Peng. Resource allocation for security services in mobile cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 191–195, April 2011.

[70] Peng Rong and Massoud Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: A markovian decision-based approach. In *Proceedings of the 40th Annual Design Automation Conference*, DAC '03, pages 906–911, New York, NY, USA, 2003. ACM.

[71] Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden. Wishbone: Profile-based partitioning for sensornet applications. In *NSDI*, volume 9, pages 395–408, 2009.

[72] Jan S. Rellermeyer, Oriana Riva, and Gustavo Alonso. Alfredo: An architecture for flexible interaction with electronic devices. In Valérie Issarny and Richard Schantz, editors, *Middleware 2008: ACM/IFIP/USENIX 9th International Middleware Conference Leuven, Belgium, December 1-5, 2008 Proceedings*, pages 22–41, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[73] J. S. Rellermeyer, M. Duller, and G. Alonso. Engineering the cloud from software modules. In *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 32–37, May 2009.

[74] E.E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. Master's thesis, Carnegie Mellon University, 2009.

[75] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3):270–284, Jun 2011.

[76] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. *SIGOPS Oper. Syst. Rev.*, 31(5):276–287, October 1997.

[77] XMPP Standards Foundation. Extensible messaging and presence protocol (xmpp). Available: https://xmpp.org/, Accessed: January 2017.

[78] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: A computation offloading framework for smartphones. In Martin Gris and Guang Yang, editors, *Mobile Computing, Applications, and Services: Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers*, pages 59–79, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[79] J. Yue, D. Zhao, and T. D. Todd. Cloud server job selection and scheduling in mobile computation offloading. In *2014 IEEE Global Communications Conference*, pages 4990–4995, Dec 2014.

[80] Q. Xia, W. Liang, Z. Xu, and B. Zhou. Online algorithms for location-aware task offloading in two-tiered mobile cloud environments. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 109–116, Dec 2014.

[81] L. Zhang, D. Fu, J. Liu, E. C. H. Ngai, and W. Zhu. On energy-efficient offloading in mobile cloud for real-time video applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):170–181, Jan 2017.

[82] X. Wang, J. Wang, X. Wang, and X. Chen. Energy and delay tradeoff for application offloading in mobile cloud computing. *IEEE Systems Journal*, 11(2):858–867, June 2017.

[83] J. Li, Z. Peng, B. Xiao, and Y. Hua. Make smartphones last a day: Pre-processing based computer vision application offloading. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 462–470, June 2015.

[84] A. Ellouze, M. Gagnaire, and A. Haddad. A mobile application offloading algorithm for mobile cloud computing. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 34–40, March 2015.

[85] A. Ellouze and M. Gagnaire. Toward an optimized servers placement for mobile applications offloading. In *2015 IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB)*, pages 1–7, Oct 2015.

[86] W. Qiu, Z. Zheng, X. Wang, X. Yang, and M. R. Lyu. Reliability-based design optimization for cloud migration. *IEEE Transactions on Services Computing*, 7(2):223–236, April 2014.

[87] K. Elgazzar, P. Martin, and H. S. Hassanein. Cloud-assisted computation offloading to support mobile services. *IEEE Transactions on Cloud Computing*, 4(3):279–292, July 2016.

[88] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi. Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2016.

[89] Shumao Ou, Kun Yang, and Jie Zhang. An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, 3(4):362 – 385, 2007. Middleware for Pervasive Computing.

[90] S. Yang, X. Bei, Y. Zhang, and Y. Ji. Application offloading based on r-osgi in mobile cloud computing. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 46–52, March 2016.

[91] Maofei Deng, Hui Tian, and Bo Fan. Fine-granularity based application offloading policy in cloud-enhanced small cell networks. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 638–643, May 2016.

[92] R. Golchay, F. Le Moult, J. Ponge, and N. Stouls. Automated application offloading through ant-inspired decision-making. In *2016 13th International Conference on New Technologies for Distributed Systems (NOTERE)*, pages 1–6, July 2016.

[93] Mohammad Goudarzi, Mehran Zamani, and Abolfazl Toroghi Haghighat. A fast hybrid multi-site computation offloading for mobile cloud computing. *Journal of Network and Computer Applications*, 80:219 – 231, 2017.

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

[95] Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424, Sep 2000.

[96] Aharon Ben-Tal and Arkadi Nemirovski. Robust convex optimization. *Mathematics of operations research*, 23(4):769–805, 1998.

[97] Aharon Ben-Tal and Arkadi Nemirovski. Robust truss topology design via semidefinite programming. *SIAM Journal on Optimization*, 7(4):991–1016, 1997.

[98] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1 – 13, 1999.

[99] Laurent El Ghaoui and Hervé Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

[100] Laurent El Ghaoui, Francois Oustry, and Hervé Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998.

[101] Dimitris Bertsimas, David B Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM review*, 53(3):464–501, 2011.

[102] Alper Atamtürk. Strong formulations of robust mixed 0–1 programming. *Mathematical Programming*, 108(2):235–250, Sep 2006.

[103] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, Sep 2003.

[104] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[105] Elodie Adida and Georgia Perakis. A robust optimization approach to dynamic pricing and inventory control with no backorders. *Mathematical Programming*, 107(1):97–129, Jun 2006.

[106] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, Mar 2004.

[107] Dimitris Bertsimas and Aurélie Thiele. A robust optimization approach to supply chain management. In Daniel Bienstock and George Nemhauser, editors, *Integer Programming and Combinatorial Optimization: 10th International IPCO Conference, New York, NY, USA, June 7-11, 2004. Proceedings*, pages 86–100, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[108] Dimitris Bertsimas and Aurélie Thiele. A robust optimization approach to inventory theory. *Operations Research*, 54(1):150–168, 2006.

[109] Aharon Ben-Tal, Tamar Margalit, and Arkadi Nemirovski. Robust modeling of multi-stage portfolio problems. In Hans Frenk, Kees Roos, Tamás Terlaky, and Shuzhong Zhang, editors, *High Performance Optimization*, pages 303–328. Springer US, Boston, MA, 2000.

[110] Gert RG Lanckriet, Laurent El Ghaoui, Chiranjib Bhattacharyya, and Michael I Jordan. A robust minimax approach to classification. *Journal of Machine Learning Research*, 3(Dec):555–582, 2002.

[111] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 56(2):82–90, February 2013.

[112] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. S2e: A platform for in-vivo multi-path analysis of software systems. *SIGPLAN Not.*, 47(4):265–278, March 2011.

[113] Peter Boonstoppel, Cristian Cadar, and Dawson Engler. Rwset: Attacking path explosion in constraint-based test generation. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 351–366. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[114] Huaming Wu, Qiushi Wang, and K. Wolter. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *2013 IEEE In-*

ternational Conference on Communications Workshops (ICC), pages 728–732, June 2013.

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

[116] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2000.

[117] Richard Ernest Bellman and Stuart E Dreyfus. *Applied dynamic programming*. Rand Corporation, 1962.

[118] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms, 3rd Edition*. The MIT press, 2009.

[119] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679 – 688, 2006.

[120] Mono Project. Mono [online]. Available: http://www.mono-project.com/, Accessed: January 2017.

[121] DigitalOcean. DigitalOcean cloud [online]. Available: http://cloud.digitalocean.com/, Accessed: January 2017.

[122] Cesar Roberto de Souza. Accord.NET Framework [online]. Available: http://accord-framework.net/, Accessed: January 2017.