Fog and Cloud Computing Optimization in Mobile IoT Environments

José Carlos Ribeiro Vieira

Thesis to obtain the Master of Science Degree in
Electrical and Computer Engineering

Supervisor(s): Prof. António Manuel Raminhos Cordeiro Grilo
               Prof. João Coelho Garcia

Examination Committee
Chairperson: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Prof. Antonio Manuel Raminhos Cordeiro Grilo
Member of the Committee: Prof. Paolo Romano

November 2019
Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
Acknowledgments

I want to thank my parents and to my sister for their support throughout the years and for giving me everything that I needed to complete this academic and personal journey.

I am grateful for the love and support of my girlfriend, Inês Ferreira, throughout this journey. We made this path together and I am really grateful for that.

I also have to thank all the people that were involved in this academic stage. In particular, I want to thank my colleague Pedro Carmo for his friendship, help and motivation in each project that we have made together.

Finally, I would also like to acknowledge and thank my supervisors Prof. António Grilo and Prof. João Garcia for their guidance, patience, motivation and wisdom throughout this project. Their help, ideas and advices were fundamental to the development of this work.

To each and every one of you - Thank you for everything.
Abstract

With the surge of ubiquitous demand on high-complexity and quality of mobile IoT services, new computing paradigms have emerged. Motivated by the long and unpredictable end-to-end communication latency experienced in cloud computing as well as the rapid growth of mobile traffic, fog computing emerges as the most comprehensive and natural paradigm to support real-time applications and to get more efficient with the data sent to the cloud. From the performed analysis, the lack of research in dynamic environments regarding both the client and fog nodes is noticeable. Moreover, most of the existing proposed schemes only consider few specific objectives for the system optimization as well as static computing resources. In the present work, a novel fog computing architecture has been designed and evaluated with the purpose of finding a solution to the aforementioned issues. A novel optimization problem formulation is also proposed in order to match the characteristics of the proposed architecture. A set of optimization algorithms were developed to solve the formulated problem. Moreover, the proposed architecture has been successfully implemented in a suitable developed simulation toolkit. The performance of the optimization algorithms was assessed in different static and mobile scenarios using the QoS offered to the clients and the system provider objectives as the main metrics. It is observed that the proposed architecture effectively helps to improve the offered QoS to its users in mobile and static environments while meeting the system provider objectives.

Keywords: Fog Computing, IoT, Mobile Environments, Multi-Objective Optimization, iFogSim
Resumo

Com o aumento da solicitação omnipresente de serviços IoT móveis de alta qualidade e complexidade, novos paradigmas de computação têm emergido. Motivado pelas longas e imprevisíveis latências de comunicação de ponta a ponta experimentadas na utilização de serviços de cloud computing, bem como pelo rápido crescimento do tráfego móvel, fog computing surge como o paradigma mais natural e abrangente para suportar aplicações em tempo real e ser mais eficiente nos dados enviados para a cloud. A partir da análise realizada, a falta de investigação em ambientes dinâmicos relativamente aos clientes e aos nós de fog é notória. Além disso, a maioria das arquiteturas anteriormente propostas apenas consideram alguns objetivos específicos na otimização do sistema, bem como recursos de computação estáticos. No presente trabalho, uma arquitetura nova de fog computing foi projetada e avaliada com o objetivo de encontrar uma solução para os problemas acima mencionados. Uma nova formulação do problema de otimização é também proposta de forma a corresponder às características da arquitetura. Um conjunto de algoritmos de otimização foi desenvolvido para resolver o problema formulado. Além disso, a arquitetura proposta foi implementada com sucesso numa ferramenta de simulação desenvolvida. O desempenho dos algoritmos de otimização foram avaliados em diferentes cenários estáticos e móveis, utilizando a qualidade de serviço oferecida aos clientes e os objetivos do operador do sistema como as principais métricas. É verificado que a arquitetura proposta ajuda efetivamente a melhorar a qualidade de serviço oferecida aos seus clientes em ambientes estáticos e móveis, enquanto atende aos objetivos do operador do sistema.

Palavras-chave: Fog Computing, IoT, Ambientes Dinâmicos, Otimização Multi-Objetivo, iFogSim
# Contents

Acknowledgments ......................................................... v
Abstract ................................................................. vii
Resumo ................................................................. ix
List of Tables ........................................................... xiii
List of Figures ........................................................... xv
Glossary ................................................................. xx

## 1 Introduction

1.1 Motivation ......................................................... 3
1.2 Objectives ......................................................... 4
1.3 Outline ........................................................... 5

## 2 Background and Related Work

2.1 Related Computing Paradigms ................................... 7
2.2 Fog Computing Architecture ................................... 10
  2.2.1 Actors ....................................................... 11
  2.2.2 Orchestration ............................................... 12
2.3 Migration Optimization in Mobile Fog Environments .......... 14
  2.3.1 QoS-Aware ................................................... 15
  2.3.2 Bandwidth-Aware .......................................... 15
  2.3.3 Energy-Aware ............................................... 16
  2.3.4 Cost-Aware .................................................. 17
  2.3.5 Multi-Objective ............................................ 19
  2.3.6 Concluding Remarks ....................................... 21
2.4 Modeling and Simulation Toolkits ............................... 24
2.5 Summary ........................................................ 29

## 3 Model Implementation

3.1 Architecture ..................................................... 31
  3.1.1 Entities ..................................................... 32
  3.1.2 Services .................................................... 37
3.2 Assumptions ..................................................... 38
3.3 Problem Formulation ................................................................. 41
  3.3.1 System and Problem Variables ............................................. 42
  3.3.2 Objectives ................................................................. 45
  3.3.3 Constraints ................................................................. 49
  3.3.4 Concluding Remarks ....................................................... 51
3.4 Simulation Architecture ............................................................ 53
  3.4.1 Scenario Design ......................................................... 53
  3.4.2 Optimization .............................................................. 57
  3.4.3 Results ................................................................. 61
3.5 Summary ............................................................................ 61

4 Results ............................................................................. 63
  4.1 Simulation Parameters ......................................................... 63
  4.2 Behaviour and Performance of the Proposed System ..................... 66
    4.2.1 Simulator, Architecture, and Problem formulation Validation .......... 67
    4.2.2 Algorithm Comparison ................................................ 70
    4.2.3 Study of the Impact of the Relative Tolerances ......................... 72
    4.2.4 Study of the Impact of the Migrations .................................. 73
  4.3 Summary ........................................................................ 74

5 Conclusions ..................................................................... 77
  5.1 Future Work ................................................................. 78

Bibliography .......................................................................... 81

A Optimization Algorithms .......................................................... 90

B Developed Graphical User Interface ............................................... 96

C Algorithm Plots .................................................................. 97

D Declaration ....................................................................... 100
List of Tables

2.1 Features of fog computing related paradigms. ................................. 9
2.2 Features comparison of the above described works. ......................... 22
2.3 Problem comparison of the above described works. .......................... 23
2.4 Comparison of fog and related computing paradigms simulators. .......... 26

3.1 System variables. ................................................................. 43
3.2 Problem variables. ............................................................... 44
3.3 Characteristics of the application edges. ...................................... 44

4.1 List of fixed simulation parameters. ........................................... 64
4.2 List of the optimization algorithm parameters used in this section. ........ 65
4.3 Application edges of the EEG Tractor Beam Game. ......................... 65
4.4 Application dependencies of the EEG Tractor Beam Game. ................ 66
4.5 Application edges of the Intelligent Surveillance System. .................. 66
4.6 Application dependencies of the Intelligent Surveillance System. ......... 67
4.7 Comparison of the algorithm and simulation results using different priority sets. 68
4.8 Performance comparison of the different proposed optimization algorithms. 72
4.9 Impact comparison of different tolerance sets using the GA with the Config. 2. 72
4.10 Migration impact under different velocities and VM sizes. ................. 75
List of Figures

2.1 Comparison of fog computing and its related computing paradigms. ......................... 9
2.2 Typical architecture of fog computing. ......................................................... 11

3.1 DAG of an application example. ................................................................. 38
3.2 Movement of a mobile node in the handover zone. ........................................... 39
3.3 DAG of another application example. .............................................................. 44
3.4 Directed graph example used to compute the shortest path between two nodes. .... 50

4.1 DAG of the EEG Tractor Beam Game. ............................................................. 65
4.2 DAG of the Intelligent Surveillance System. ..................................................... 66
4.3 First scenario: a static fog environment composed by two interconnected fog, one cloud and one client nodes. ................................................................. 67
4.4 Second scenario: a static fog environment composed by three interconnected fog, one cloud and two client nodes. ............................................................... 69
4.5 Third scenario: a static fog environment composed by four interconnected fog, one cloud and three client nodes. ............................................................... 70
4.6 Dynamic environment composed by a cloud and two static and interconnected fog nodes, as well as a mobile fog and two client nodes. .............................................. 73

B.1 Developed graphical user interface. ............................................................... 96

C.1 Iteration-value plot for the first scenario with priority set 1. ......................... 97
C.2 Iteration-value plot for the first scenario with priority set 2. ......................... 98
C.3 Iteration-value plot for the first scenario with priority set 3. ......................... 98
C.4 Iteration-value plot for the first scenario with priority set 4. ......................... 99

D.1 Declaration. ......................................................................................... 100
Glossary

ADMM Alternating Direction Method of Multipliers
AP Access Point
BFA Brute Force Algorithm
BSPP Basic Service Placement Problem
cC Cloudlet Computing
CC Cloud Computing
CEP Complex Event Processing
CLOUDS CLOUid computing and Distributed Systems
CPU Central Processing Unit
CSP Cloud Service Providers
DAG Directed Acyclic Graph
DDF Distributed DataFlow
DSP Digital Signal Processors
EC Edge Computing
FIFO First In, First Out
FOV Fields Of View
FPGA Field Programmable Gate Array
GA Genetic Algorithm
GBD Generalized Benders Decomposition
GPU Graphics Processing Unit
GUI Graphical User Interface
IoT Internet of Things
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPM</td>
<td>Interior-Point Method</td>
</tr>
<tr>
<td>ITEM</td>
<td>Iterative Expansion Moves</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LBS</td>
<td>Location-Based Service</td>
</tr>
<tr>
<td>LOTEC</td>
<td>Lyapunov Optimization on Time and Energy Cost</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>LTE</td>
<td>Long-Term Evolution</td>
</tr>
<tr>
<td>MACC</td>
<td>Mobile Ad hoc Cloud Computing</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad hoc NETwork</td>
</tr>
<tr>
<td>MC</td>
<td>Mobile Computing</td>
</tr>
<tr>
<td>MCAPP</td>
<td>Multi-Component Application Placement Problem</td>
</tr>
<tr>
<td>MCC</td>
<td>Mobile Cloud Computing</td>
</tr>
<tr>
<td>MCEP</td>
<td>Mobile Complex Event Processing</td>
</tr>
<tr>
<td>MI</td>
<td>Million Instructions</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed-Integer Linear Programming</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed-Integer NonLinear Programming</td>
</tr>
<tr>
<td>MIPS</td>
<td>Million Instructions Per Second</td>
</tr>
<tr>
<td>MOERA</td>
<td>Mobility-agnostic Online Edge Resource Allocation</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Cards</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PAYG</td>
<td>Pay As You Go</td>
</tr>
<tr>
<td>PRIMAL</td>
<td>PRofIt Maximization Avatar pLacement</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>PTZ</td>
<td>Pan–Tilt–Zoom</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RA</td>
<td>Random search Algorithm</td>
</tr>
</tbody>
</table>
**RSS**  Received Signal Strength

**UAV**  Unmanned Aerial Vehicles

**UE**  User Equipment

**VM**  Virtual Machine

**VR**  Virtual Reality

**WWS**  Writable Working Set
Chapter 1

Introduction

World is growing at a fast pace and so is data. Agility and flexibility of big data applications are gradually taking the form of the Internet of Things (IoT), things featuring unique identities, being accessible from anywhere in the world. Ubiquitous deployment of interconnected devices is estimated to reach 50 billion units by 2020 [1]. This exponential growth is broadly supported by the increasing number of mobile devices (e.g., smart phones, tablets), smart sensors serving different markets (e.g., autonomous transportation, industrial controls, wearables), wireless sensors and actuators networks. The number of mobile devices are predicted to reach 11.6 billion by 2021, exceeding the world’s projected population at that time (7.8 billion), where the subset of wearables is expected to be 929 million [2].

Managing the data generated by IoT sensors and actuators is one of the biggest challenges faced when deploying an IoT system. Although this kind of device has evolved radically in the last years, battery life, computation and storage capacity remain limited. This means that they are not suitable for running heavy applications, being necessary, in this case, to resort to more powerful computing resources, likely owned by third parties.

Cloud Computing (CC) is a resource-rich environment that has been imperative in expanding the reach and capabilities of IoT devices. It enables clients to outsource the allocation and management of resources (hardware or software) to the cloud. In addition, to avoid over- or under-provisioning, Cloud Service Providers (CSPs) also afford dynamic resources for a scalable workload, applying a Pay As You Go (PAYG) cost model. This way, cloud computing is the on-demand delivery of computing power, database storage, applications, and other IT resources through a cloud services platform via the Internet with PAYG pricing [3]. Besides, it also brings other advantages such as availability, flexibility, scalability, reliability, to mention a few.

Despite the benefits of cloud computing, there are two main problems linked to IoT applications, which remain unresolved. The first and the most obvious, is the fact that cloud servers reside in remote data centers. Consequently, the end-to-end communication may be subject to long and unpredictable delays (characteristic of multi-hop transmissions over the Internet). Some applications, with stringent latency requirements, cannot support such delays. Augmented reality applications, which use head-tracked systems, for example, require end-to-end latencies lower than 16 ms [4]. Cloud-based virtual
desktop applications require end-to-end latency below 60 ms if they are to match Quality of Service (QoS) of local execution [5]. Remotely rendered video conference, on the other hand, demand end-to-end latency below 150 ms [6]. On the other hand, the exponential growing number of IoT devices raises the second problem: as the number of connected devices increases, the bandwidth required to support them becomes too large for centralized processing (i.e., CC). To overcome these drawbacks, there are two likely solutions: (1) to increase the number of centralized cloud data centers, which will be too costly, and (2) to get more efficient with the data sent to the cloud.

A solution which has already been proposed is to bring the cloud closer to the end devices, where entities such as base-stations would host smaller sized clouds. This idea has brought the emergence of several computing paradigms such as cloudlets [7], fog computing [8], edge computing [9], and follow me cloud [10], to name a few. These are different solutions, often confused in the literature, which provide faster approaches and gain better situational awareness in a more timely manner. Regardless of their characteristics, they all share the same goal, implementing solution (2) (i.e., to get more efficient with the data sent to the cloud).

As it will be discussed later, fog computing, also known as fog networking or fogging, is the most comprehensive and natural paradigm to get more efficient with the data sent to the cloud. A simple definition of fog is “cloud closer to the ground”, which gives an idea of its functioning. Fog is thus a decentralized computing infrastructure that aims to enable computing, storage, networking, and data management not only in the cloud, but also along the cloud-to-thing path as data traverses the network towards the cloud. Essentially, it extends cloud computing and services along the network itself, bringing them closer to where data is created and acted upon. Fog brought these services closer to the end devices due to its low hardware footprint and low power consumption. This way, both problems raised by the use of cloud computing, may be solved, or at least significantly mitigated. First, the path travelled by the data in the sense-process-actuate model is much shorter (ideally, just one hop to send data and another to receive the results), allowing latency to be much smaller compared to the traditional cloud computing. Second, through geographical distribution, there are significant amounts of data that are no longer travelling up to the cloud. Fog computing, prefers to process data, as much as possible, in the nodes closer to the edge of the network. In a simplistic manner, it only considers transferring the data further if there is not enough computational power to meet the demands.

Nevertheless, cloud is still more suitable than fog for massive data processing when the latency constraints are not so tight. Therefore, even though fog computing has been proposed to grant support for IoT applications, it does not replace the needs of cloud-based services. In fact, fog and cloud complement each other. Together, they offer services even further optimized to IoT applications. It should be noted that Internet connectivity is not essential for the fog-based services to work, which means that services can work independently and send necessary updates to the cloud whenever the connection is available [11].

This cooperation between fog and cloud services has a widely range of use cases. For instance, drones, also known as Unmanned Aerial Vehicles (UAVs), can be employed in plenty of applications which are difficult or dangerous for humans (e.g., military, disaster or emergency missions). The mode
of operation of drones has progressively changed from the traditional remote control by humans to a sophisticated autonomous control, able to operate without human intervention. However, as all processing required to analyze the collected data and make decisions are performed on-board, the flight autonomy decreases substantially. For instance, let us consider an extremely resource intensive application, say search for missing persons in disaster zones using image processing. Fog computing, being able to provide real-time responses, is capable of hosting and process the autonomous control module, and send its output results to the drone. With regard to the image processing module, as it has no such tight time constraints, it can be forwarded to the cloud, which will be responsible for hosting, and processing the collected data (e.g., video, images or heat sensor data) and give its output to the person in charge of the rescue mission. This way, the required processing operations on-board are tremendously reduced, resulting in an increase in terms of the overall duration of the flight. Note that, its up to the orchestrator, based on its objectives, to decide whether to place the modules in the fog or in the cloud, provided that constraints (e.g., time boundaries) are fulfilled.

Nonetheless, in order to support such a wide rage of applications, and similarly to cloud computing, fog can use the concept of virtualization to grant heterogeneity. IoT applications may span many Operating Systems (OSs) and application environments (e.g., Android, iOS, Linux, Windows), as well as diverse approaches to partitioning and offloading computation. There is churn in this space from new OS versions, patches to existing OS versions, new libraries, new language run-time systems, and so on. In order for fog to support all these variants, a level of abstraction can be introduced, which cleanly encapsulates this complexity in a Virtual Machine (VM) or a container. Moreover, it enables applications to coexist in a physical server (host) to share resources. Meanwhile, in fog computing, the use of virtualization techniques is also crucial to support migration of applications. The environment of fog computing is, by nature, unquestionably dynamic due to numerous factors such as: the mobility of nodes (i.e., clients and servers), changes in the network, number of applications, requests, nodes, etc. In this fashion, in order to guarantee the desired QoS, changes in the placement of applications may have to be performed through the migration of VMs or containers. Summing up, in this context, virtualization is a vital technology at different levels, namely: (1) isolation between untrusted user-level computations, (2) mechanisms for authentication and access control, (3) dynamic resource allocation for user-level computations, (4) ability to support a wide range of user-level computations with minimal restrictions on their process structure, programming languages or OSs, (5) mobility, migration and task offloading mechanisms, (6) power efficiency, and (7) fault tolerance [12].

1.1 Motivation

Despite the benefits that fog promises to offer, such as low latency, heterogeneity, scalability and mobility, the current model suffers from some limitations that must be overcome.

There is lack of support for mobile fog computing. Most of the existing literature assumes that the fog nodes are fixed, or only considers the mobility of IoT devices [11]. Less attention has been paid to mobile fog computing and how it can improve the QoS, cost, and energy consumption. For instance,
a bus or a train could have computational power; as a fog node, it could provide offloading support to both end devices (inside and outside it) and other fog servers. The same could be applied to cars that are nowadays getting increasingly better in terms of computational power. Both would be extremely useful to enhance the resources and capabilities of fog computing. Specially in environments such as large urban areas, where traffic congestion is frequent or when those are parked (e.g., while an electric vehicle is charging). On top of that, it would reduce the implementation costs since it would no longer require such computational power in the fixed fog nodes. Finally, it would reduce the costs to the client both in terms of latency and energy consumption, since the fog nodes to which they are connected may be even closer.

Another limitation of fog computing is to take into account few parameters in the decision-making of migration. Most of the existing schemes that are proposed for fog systems, such as offloading, load balancing, or service provisioning, only consider few objectives (e.g., QoS, cost) and assume other objectives do not affect the problem [11]. Fog servers are less powerful than clouds due to the high deployment cost. If many requests are made to the same fog node at the same time, it will not have enough computational and memory capabilities to give a prompt response. So, it raises the question: should a service currently running in one fog node be migrated to another one, and if yes, where? While conceptually simple, it is challenging to make these decisions in an optimal manner. Offloading tasks to the closer available server seems to be the solution, however, to migrate either the VM or the container that was initially one-hop away from the IoT device to a multi-hop away server, will increase the network distance. Consequently, it raises the end-to-end latency and the bandwidth usage by the intermediate links. Besides, this decision still has to take into account the cost for both the client (e.g., migration time, computational delay) and the provider (e.g., computing and migration energy). Ignoring some of these variables can lead to wrong decisions that will violate latency constraints of users’ applications.

1.2 Objectives

This work intends to tackle two of the current limitations whose treatment is lacking in the literature related with fog computing placement optimization. One is to provide mobility support in fog computing environments, not exclusively to the end devices but also to the fog nodes; the other other is to achieve multi-objective fog system design. These two main objectives can be decomposed into the following objectives:

- Analyze the current mobility approaches in fog computing, with respect to IoT or fog nodes;
- Analyze several optimization algorithms adopted in the field of data placement;
- Analyze and compare different currently available simulation toolkits in the fields of fog and related computing paradigms;
- Propose a novel architecture allowing the mobility of IoT and fog nodes while preserving user QoS through migration mechanisms;
• Propose a novel problem formulation allowing to optimize the system performance;

• Propose different decision-making algorithms used to optimize the formulated objectives to serve as term of comparison;

• Development of the proposed architecture in a suitable simulation toolkit;

• Performance evaluation of the system in the simulation toolkit under selected QoS metrics.

1.3 Outline

The remainder of the document is structured as follows. Chapter 2 presents a background section, which investigates the different computing concepts and shows why fog computing appears as the most comprehensive and natural paradigm to get more efficient with the data sent to the cloud. It also introduces a detailed fog computing architecture, including its actors and its different proposed orchestration mechanisms. This section is followed by the state-of-the-art and a review of relevant works in the context of the objectives described above. Moreover, it also presents and discusses the different available simulators in fog and related computing paradigms field. Chapter 3 describes an overview of the proposed architecture and the main design choices of the developed system. The proposed problem formulation in order to optimize the system performance and the different proposed decision-making algorithms are also presented. Finally, it also exhibits the simulation model developed used to implement different scenarios and evaluate the system performance while using different decision-making algorithms. Chapter 4 provides a comparative evaluation of the performance of the developed algorithms relative to the considered application scenarios. This is done by resorting to selected QoS metrics collected by means of multiple computer simulations. Finally, Chapter 5 concludes the document by summarizing the achieved results, considering the initial objectives. Some future directions to improve the work are also given.
Chapter 2

Background and Related Work

The solution proposed in this document leverages knowledge obtained from studying several concepts and systems from the current state-of-the-art. In this chapter, an overview of those concepts and systems will be given, stating for each of them their advantages and disadvantages. This chapter is structured as follows. Section 2.1 presents different methods to push intelligence and computing power closer to the source of the data and why this work adopted fog computing for this purpose. Section 2.2 describes the generic fog computing architecture, its actors and the different orchestration approaches. Section 2.3 discusses several optimization algorithms regarding migration of virtual resources (e.g., VMs). Based on the discussion and comparison of the reviewed works, there are highlighted the set of features that the novel proposed architecture aims to cover. Finally, Section 2.4 shows and discusses several open-source simulators of fog-related computing paradigms along with their characteristics.

2.1 Related Computing Paradigms

In what concerns fog computing standardization, there is a lack of unanimity. Fog has been variously termed as cloudlets, edge computing, etc. Different research teams are proposing many independent definitions of fog. As there is a research gap in the definitions and standards for fog computing, this work follows the definitions that Ashkan Yousefpour et al. [11] present. This section presents some paradigms that were proposed in order to bring cloud closer to the end devices, and discusses their pros and cons. As a conclusion, it is shown why fog computing is the natural platform for IoT.

Mobile Computing (MC) is characterized by the processing being performed by mobile devices (e.g., laptops, tablets, or mobile phones). It is required to overcome the inherent limitations of environments where connectivity is sparse or intermittent and where there is low computing power. As this model only uses mobile devices to provide services to clients, there is no need for extra hardware. They already have built-in communication modules such as Bluetooth, WiFi, ZigBee, etc. The disadvantages of MC are mainly due to its hardware nature (i.e., low resources, balancing between autonomy and interdependence and the need for mobile clients to efficiently adapt to changing environments) [13]. This restricts the applications where this paradigm is feasible.
Cloud computing, is a key element to validate the importance of MC. The interaction between them results in a new paradigm, called Mobile Cloud Computing (MCC). With MCC, applications can be partitioned at run-time and computationally intensive components can be offloaded from mobile devices to the cloud [14]. This way, MCC has high availability of computing resources, scaling the type of applications where it is possible to use. Also, this characteristic increases the autonomy of mobile devices and enables a much broader range of mobile subscribers, rather than the previous laptops, tablets, or mobile phones. As MCC relies on cloud-based services, it suffers from the inherent communication latency disadvantage, which makes it unsuitable for some delay-sensitive applications.

In order to overcome the inherent need of MCC for an infrastructure or a centralized cloud, Mobile Ad hoc Cloud Computing (MACC) was proposed [15]. MACC is a Mobile Ad hoc NETwork (MANET), consisting of a set of mobile ad hoc nodes that form a dynamic and temporary network enabled by routing and transport protocols. This computing paradigm allows to form local clouds which can be used for networking, storage and computation. Although MACC do not rely on external cloud-based services, which mitigates the latency problem, power consumption constraints are a major concern. Moreover, the formed temporary local cloud may still be computationally weak, and as there is no infrastructure, mobile devices are also responsible for routing traffic among themselves.

Open Edge Computing defines Edge Computing (EC) as a computation paradigm that provides small data centers (edge nodes) in proximity to the users, enabling a dramatic improvement in customer experience through low latency interaction with compute and storage resources just one hop away from the user [16]. OpenFog Consortium states that fog computing is often erroneously called edge computing, but there are key differences between the two concepts [17]. Although they have similar concepts, edge computing tends to be limited to the edge devices (i.e., located in the IoT node network), excluding the cloud from its architecture. Whereas, fog computing is hierarchical and it is not limited to a local network, but instead it provides services anywhere from cloud to things.

Cloudlet Computing (cC) is another mobile computing paradigm which aims to bring cloud closer to end devices through the use of cloudlets. It extends MCC by adding the cloudlet tier to its architecture. This way, as Y. Jararweh et al. [18] propose, cloudlet is the middle tier of a 3-tier continuum: mobile-cloudlet-cloud. Cloudlet is a smaller sized cloud (typically one hop away) with lower computational capacity in which mobile users can exploit their VM to rapidly instantiate customized-service software in a thin client fashion. Through those VMs, cloudlets are capable of providing resources to end devices in real-time over a WLAN or cellular network. The relatively low hardware footprint, results in moderate computing resources, but lower latency and energy consumption and higher bandwidth compared to cloud computing. Despite the clear benefits of cC, fog computing offers a more generic alternative for not being limited solely in this 3-tier.

Mist computing emerges to push IoT analytics to the “extreme edge” [19]. This computing paradigm is an even more dispersed version of fog. Mist computing layer is composed by mist nodes that are perceived as lightweight fog nodes. They are more specialized and dedicated nodes with low computational resources (e.g., microcomputers, microcontrollers) that are even closer to the end devices than the fog nodes [20]. Therefore, mist computing can be seen as the first (non-mandatory) layer in the IoT-fog-
cloud continuum. It extends compute, storage, and networking across the fog through the things. This decreases latency and increases subsystems’ autonomy. It can be implemented in order to enhance the services of predominance of wireless access and mobility support.

Although there exist other similar computing paradigms, this state-of-the-art section had as first objective to investigate the most addressed concepts in the literature. The purpose was to understand their characteristics and to identify current limitations that must be tackled by novel solutions, in order to allow the deployment of delay-sensitive IoT systems in mobile environments. Figure 2.1 compares fog computing and its related computing paradigms in terms of their location and distance from the core clouds and Table 2.1 compares the features of the paradigms described above.

These computing paradigms present different pros and cons, having been proposed to cover different use cases. Even so, fog computing is suited for many use cases, including data-driven computing and low-latency applications, being the most versatile and comprehensive one. Fog is flexible enough to interact and take advantage of other paradigms such as edge, cloud, cloudlet and mist computing.
Nonetheless, it may not be suitable for a few extreme use cases, such as disaster recovery or sparse network topologies where ad hoc computing (e.g., MACC) may be a better fit.

### 2.2 Fog Computing Architecture

Fog computing is a great resource to support IoT applications’ requirements in mobile environments. Taking into account what has been mentioned in Chapter 1 and Section 2.1, it has the following fundamental characteristics which validate the statement mentioned above (refer to Table 2.1):

- **Heterogeneity support.** Supports processing, over different type of nodes, of data collected from different actors acquired through multiple types of network communication, wide diversity applications and services;

- **Geographical distribution.** In contrast to the centralized cloud, uses anything between the cloud and *things* to provide ubiquitous computing, allowing continuity of service in mobile environments;

- **Contextual location awareness, and low latency.** Provides low latency due to the proximity between the IoT devices and the fog nodes. Also, the contextual location allows them to be aware of the cost of communication latency with both other fog nodes and the end devices, allowing the distribution of applications across the network to be organized in a weighted manner;

- **Mobility support.** Uses its heterogeneity support, geographical distribution and contextual location awareness to support mobile devices to use its services;

- **Real-time interactions.** In contrast to the batch processing performed by cloud servers, fog uses its characteristics to support the deployment and processing of real-time applications;

- **Scalability and agility of federated, fog-node clusters.** Fog is adaptive; may form clusters-of-nodes or cluster-of-clusters to support elastic compute, resource pooling, etc., supporting large-scale applications;

- **Multiple IoT applications.** Fog devices handle multiple IoT applications competing for their limited resources;

- **Virtualization support.** Introduces a software abstraction between the hardware and the OS and application running on the hardware;

- **Interoperability and federation.** Uses cooperation of different providers to support heavy applications such as real-time streaming. Moreover, it supports migration of applications to more suited fog servers depending on the current context;

- **Predominance of wireless access.** Uses any type of connections to support connectivity to a wide range of end devices (which may not support wired connections).
Nonetheless, as stated in Section 1.2, fog still has some limitations. In order to tackle those, its overall architecture must be understood. This includes knowing what are the actors and how they interact, how IoT nodes connect to the fog servers, how clients outsource the allocation and management of resources that they rely upon to these servers, how migration is performed, etc.

2.2.1 Actors

Figure 2.2 shows the typical fog computing architecture. As stated before the presence of cloud servers is not imperative, however it is very important for numerous applications.

Fog computing layer is composed by fog nodes/servers, which allow the deployment of distributed, latency-aware applications and services. Those nodes can be either physical (e.g., gateways, switches, routers, servers) or virtual (e.g., virtualized switches, virtual machines, cloudlets) components which provide computing resources to the connected end devices. When needed, they also provide network connectivity to centralized services (i.e., cloud). Moreover, fog nodes can operate in a centralized or decentralized manner or even be configured as stand-alone nodes.

Fog nodes are of most value in scenarios where data needs to be collected at the edge and where data from thousands or even millions of devices is analyzed and acted upon in micro and milliseconds [17]. In order to being able to support such large number of requests, especially those engaged in enhanced analytics, fog nodes may be equipped with additional hardware. Accelerator modules (refer to Fig. 2.2) can be implemented to provide supplementary computational throughput. For instance, hardware accelerators can be performed through Graphics Processing Units (GPUs); they are an optimal choice for applications that support parallelism or for stream processing. Also, fog nodes may be equipped with Field Programmable Gate Arrays (FPGAs) or even Digital Signal Processors (DSPs) for...
It is worth noting that, once fog nodes can be anything with computational and storage power in the cloud-to-things continuum, the links formed in these architectures (i.e., End device-to-Fog, Fog-to-Fog and Fog-to-Cloud) can be of any type. For instance, end devices can be connected to fog servers by wireless access technologies (e.g., WLAN, WiFi, 3G, 4G, ZigBee, Bluetooth) or wired connection. Moreover, fog nodes can be interconnected by wired or wireless communication technologies, and they can be linked into the cloud by core network using fiber transmission with low-latency.

In this architecture, the connected sensors located at the edge, generate data that can adopt two models. First, in a sense-process-actuate model, the information collected is transmitted as data streams, which is acted upon by applications running on fog devices and the resultant commands are sent to actuators. In this model, the raw data collected often does not need to be transferred to the cloud; data can be processed, filtered, or aggregated in fog nodes, producing reduced data sets. The result can then be either stored inside fog nodes or actuated upon through the actuators. Second, in a stream-processing model, sensors also send data streams, where the information mined (from the incoming streams) is stored in data centers for large-scale and long-term analytics. In this case, big data needs to be stored and does not have significant latency constraints. Being fog servers less powerful than the cloud ones, cloud is far more suited for this kind of operations. Yet, fog servers can still shrink data, doing some intermediate processing as in the previous model. This meets the aforementioned statement - although cloud is not always essential for the functioning of fog, in some applications it is beneficial or even crucial.

The applications deployed by the connected end users into the fog nodes can be treated either as a whole or as a Distributed Dataflow (DDF) programming model, in which the applications are structured as a collection of modules. N. Giang et al. [21] propose a DDF for IoT applications which use computing infrastructures across the fog and the cloud, allowing the application flow to be deployed on multiple physical devices rather than one. This can be particular useful to deploy less restricted modules in terms of latency to the upper fog layers (ideally to the cloud), leaving the fog nodes in the lower layers less overloaded, being able to respond faster to modules within tighter latency bounds. As already mentioned, fog utilizes virtualization mechanisms due to the numerous advantages offered. Hence, hosting an application involves creating a set of VMs or execution containers (e.g., Docker) and assign them to a set of physical or virtual components along the cloud-to-things continuum.

2.2.2 Orchestration

When an end device needs to offload some work to a third party, it needs somehow to know where to outsource the allocation and management of resources. To do so, this architecture also needs a discovery service which concerns in finding the best available fog server, given certain capabilities and requirements. In this context, J. Gedeon et al. [22], propose a brokering mechanism in which available surrogates (i.e., fog nodes) advertise themselves to the broker. When it receives client requests, considering a number of attributes such as network information, hardware capabilities, and distance, it finds
the best available surrogate for the client.

Finally, fog also needs an orchestration layer to monitor the current context in order to be able to take management decisions with regard to applications and data placement. In this context, C. Guerrero et al. [23] state that most of the literature considers the existence of a central broker or orchestrator gathering all system information (i.e., monitoring fog devices, clients, cloud and services), leading to poor scalability and high orchestration algorithm complexity when the number of elements is high (e.g., in smart cities). To overcome this bottleneck, O. Skarlat et al. [24, 25] consider the concept of fog colonies. Each colony has an orchestrator and an arbitrary number of fog cells (a software component running on fog nodes). These fog cells are responsible for receiving tasks from IoT devices, and depending on the available resources, decide whether to execute it in the current fog node or to transfer it to the orchestrator. This top hierarchic element (inside a colony) will then migrate this task to the cloud or to another fog colony, if the current one is not able to handle it. The authors also state that these orchestrators could be replaced applying a decentralized approach. However, it would lead to extensive coordination and voting between the involved fog cells. On the other hand, F. Bonomi et al. [26] propose a distributed orchestration. This is performed implementing a Foglet software agent in each fog node. It has small footprint yet capable of monitoring machine’s health and state. This information is then pushed to the distributed and persistent storage for global processing. The distributed database is also responsible for storing business policies defined by the fog administrators. The distributed policy is then embedded in every Foglet. Specifically, when a Foglet receives a request, it will gather policies from the policy repository and information relative to the currently active service instances from the services directory. With these informations, it tries to find an instance that satisfies the defined constraints. If such is found, it forwards the request, otherwise a new instance needs to be created.

Upon the decision to migrate some application/module, service continuity is an important parameter once downtime may degrade the perceived QoS by the end user. To perform this operation, the exploited technologies are the VMs and containers. For instance, VM synthesis reduces the image size by splitting it into multiple layers, transferring only the application-specific layer which includes both the static binary program and the runtime memory data [27], however this can still involve hundreds of megabytes. Also, live migration was introduced to reduce the downtime from the traditional non-live migration. Non-live migration, suspends the VM, transfers all the content from one physical machine to another and resumes it only after the process, continuing from the same state as before suspension. In order to perform live migration there are two different approaches. On the one hand, the pre-copy memory migration firstly transfers VM’s memory state. Meanwhile, the VM keeps running. If a page gets modified (dirty), it will be re-sent. This keeps going until either a small, Writable Working Set (WWS) has been identified, or a given number of iterations is reached. Then, the VM is suspended and sent along with the remaining dirtied pages to the target machine. On the other hand, post-copy creates and sends a snapshot of the VM state from the source physical machine to the destination, being launched at its completion. Meanwhile, during the transfer, the VM is still running in the source machine. Upon copy completion, the memory state that was kept changing is copied on demand (by page-faults) from the source to the destination machine, reducing the downtime services [28]. Too many page-faults may degrade the
performance of applications running inside the VM, thus pre-paging can also be used. It ensures that the next pages to be sent to the destination machine are pages in the vicinity of the last fault. Moreover, memory access patterns may also be implemented to further enhance this mechanism.

Despite these advances, this process can be impractical in mobile fog environments due to its large size. Container represents a lighter virtualization technique. It allows developers to package up an application with all the parts needed (e.g., libraries and other dependencies). These applications share the OS of the physical machine and some libraries and/or binaries, allowing container’s size to be much smaller. This eases both migration and hosting applications (i.e., more applications in a single machine and does not require restarting the OS upon migration) [29].

Nonetheless, there are two different approaches that can be exploited and implemented into the migration decisions. On the one hand, the reactive approach, as the name suggests, only performs migration when it is needed. When users and fog nodes become out of range, migration is performed to ensure QoS in mobile environments. However, I. Farris et al. [30] argue that downtime is not the only degrading factor of service continuity, but also the overall migration which impacts Quality of Experience (QoE) of users. On the other hand, the proactive approach deploys replicas of the user service in neighboring fog nodes (e.g., using mobility patterns). Also, periodically the state in these nodes is updated. The main goal is to reduce the migration time and improve QoE. However, this approach brings new costs at different levels such as computation, memory, and networking.

Fog servers can provide reduced latencies and help in avoiding/reducing traffic congestion in the network core. However, this comes at a price: more complex and sophisticated resource management mechanisms are needed. This raises new challenges to be overcome such as dynamically deciding when, and where (device/fog/cloud) to carry out processing of requests to meet their QoS requirements. Furthermore, in mobile environments such mechanisms must incorporate mobility (i.e., location) of data sources, sinks and fog servers in the resource management and allocation process policies to promote and take advantage of proximity between fog and end devices.

2.3 Migration Optimization in Mobile Fog Environments

When an IoT device needs to offload some application to a third party, ideally it will be connected to the nearest server, securing a hop away fog server to ensure the shortest network delay. However, as their physical distance increases either by device or server movement, their network distance (i.e., the number of hops) will also increase. Hence, both latency and bandwidth usage by the intermediate links will increase, resulting in poor connectivity. In this way, in such dynamic environments the decision-making of where to offload the work is a major concern. Moreover, even if both clients and servers are static, the end-to-end latency may increase due to unexpected crowds of mobile clients seeking to connect or making requests to the same fog server simultaneously, which may lead to QoS violations. Whenever it is justified the system needs to be readjusted. This is performed through the exchange of VMs or containers (containing the applications or modules) between fog nodes. For this reason, it is necessary to answer the following questions: When is this exchange justified? And what is the
As stated in Section 1.2, this work intends to implement multi-objective management decision-making in a novel architecture. Hence, this state-of-the-art section intends to study some proposed mechanisms in the literature.

2.3.1 QoS-Aware

The first objective that fog computing has to guarantee is QoS. When users outsource some delay-constrained task or application, they expect fog to be adaptive enough so that they can move while their time boundaries are met. Without this objective fog computing is useless once it appears, in part, to help cloud computing to overcome this limitation.

In this context, the work performed by T. Rodrigues [31] et al. is focused on increasing the QoS offered to its users by lowering both transmission and processing delays. Their goal is achieved by finding the best placement of each mobile user’s VM. They assume that each user is connected to the cloudlet which offers the best Received Signal Strength (RSS), that, in turn, is also responsible for hosting its VM. Therefore, their goal is to compute the optimal transmitting power for each cloudlet which will control the RSS and, consequently, change user connections. In order to optimize the formulated problem, a Particle Swarm Optimization (PSO) model is applied. Their architecture assumes the presence of a central unit which is responsible for collecting the physical locations of users and cloudlets and then to execute the model. It is worth noting that when any user changes its connection (i.e., connects to another cloudlet), its VM is also migrated, however this delay is not considered. Also, their work considers that each VM task arrival rate follows a Poisson process with the same rate for all users and does not specify what is the algorithm’s optimal frequency of execution.

The study performed by X. Sun et al. [32] presents a case scenario where the end devices are mobile. To perform this work they use a cloudlet network architecture to bring the computing resources from the centralized cloud to the edge. They present the PROfft Maximization Avatar pLacement (PRIMAL) strategy. PRIMAL maximizes the trade-off between the migration gain (i.e., the end-to-end delay reduction) and the migration cost (i.e., the migration overheads incurred in the avatar, which compromise its performance), by migrating the avatars (a software clone located in a cloudlet) to their optimal locations, using pre-copy live migration. To solve the formulated problem, they use the Mixed-Integer Quadratic Programming tool in the CPLEX solver to find the heuristic solution of PRIMAL. It is worth noting that the considered gain only considers the end-to-end delay reduction between the user’s base station and user’s avatar, ignoring the fact that both servers and network states will be affected by the migration of avatars (which, in turn, affects the gain and the perceived QoS). Similarly to the gain, the defined cost also does not take into consideration the servers and the network states.

2.3.2 Bandwidth-Aware

Minimization of network utilization is one of the main objectives of fog computing. In fact, fog appears to overcome this inherent limitation of cloud computing. Thus, aside from ensuring QoS, it is also important to reduce bandwidth usage. This utilization of network is essentially due to three factors: the transmis-
sion of virtualized resources (VMs or containers) which contain the applications/modules, transmission of data between the end device and the deployed application into the fog nodes, and control messages exchanged between fog nodes. If the applications are deployed using DDF programming model a fourth factor rises, the data transmission between modules. In this section, the reviewed literature propose models to mitigate bandwidth usage providing long-term QoS, reducing the number of migrations.

B. Ottenwälder et al. [33] consider an environment with mobile devices and fixed fog nodes, where users offload real-time applications such as Complex Event Processing (CEP). CEP is a paradigm where changes in sensor measurements are modeled as events, while the application is modeled as a set of event-driven operators. They state that each migration comes with a cost, consequence of the local state that also needs to be migrated along with the operators. Thus, frequent migration would significantly decrease the system performance. To overcome this limitation, they propose a placement and migration method for fog providers to support operator migrations in Mobile Complex Event Processing (MCEP) systems. Their method plans the migration ahead of time through knowledge of the MCEP system and predicted mobility patterns towards ensuring application-defined end-to-end latency restrictions and reducing the network utilization. These predicted mobility patterns were captured using three different methods: uncertain locations from the dead reckoning approach (linear), certain locations that could stem from a navigation system (navi), and learned transitions between leaf broker (learned). This method allows a minimization of migration costs by selecting migration targets that ensure a low expected network utilization for a sufficiently long time.

Also in this context, W. Zhang et al. [34] state that previous studies have proposed a static distance-based Markov Decision Process (MDP) for optimizing migration decisions. However, these models fail to consider dynamic network and server states in migration decisions, assuming that all the important variables are known. Moreover, they also point out another unaddressed problem which lies in the recalculation time interval of the method. Since running MDP is a heavy computing task, a short recalculation interval introduces a considerable overhead to the server. On the other hand, a long recalculation interval may translate into lazy migration, resulting in periods of transgression of QoS guarantees. In order to overcome these issues, the authors propose SEGUE. This model achieves optimal migration decisions by providing a long-term optimal QoS to mobile users in the presence of link quality and server load variation. Additionally, SEGUE adopts a QoS aware scheme to activate the MDP model. In other words, it only activates the MDP model when QoS violation is predicted. Thus, it avoids unnecessary migration costs and bypasses any possible QoS violations while keeping a reasonable low overhead in the servers. The problem is then formulated as a cost-reward between the predicted long term QoS improvement and the service downtime.

2.3.3 Energy-Aware

In order to achieve the QoS objective, the placement of applications and their modules has often to be moved between different entities that compose the things-fog-cloud architecture which evolves energetic costs (both in terms of processing and communication). For instance, it is needed to exchange control
messages, communicate between modules placed at different nodes, change the module placement, etc. Thus, energy awareness must be an important factor to be taken into account in the decision making algorithm of when and where to offload work to another entity in order to minimize fog infrastructure providers’ cost.

In this context, R. Deng et al. [35] focused on investigating system power consumption and network delay trade-off in cloud-fog services. They formulate a workload allocation problem, which suggests the optimal workload allocations between fog and cloud toward the minimal power consumption with the constrained service delay. This was performed through the modeled power consumption and delay functions of each part of the fog-cloud computing system. It is worth noting that power consumption only considers energy consumption of work computation, disregarding communication costs. The problem is then tackled using an approximate approach through decomposition, and formulation of three subproblems, being solved through existing optimization techniques, including the Generalized Benders Decomposition (GBD), Hungarian algorithm, and Interior-Point Method (IPM). This work also does not considers dynamic environments. All variables are static including the position of fog nodes and end devices. Also there is no cooperation between fog nodes and the communication delay between a fog node and a cloud server is only characterized by its latency, ignoring the bandwidth. Similarly to the majority of the presented works, the decision-making is performed in a centralized manner.

Y. Xiao et al. [36] investigate two performance metrics for fog computing networks: the QoS of mobile users and the power efficiency of fog nodes. In their scheme, fog nodes can process or offload to other fog nodes part of the workload that was initially sent to the cloud. Fog nodes decide whether to offload the workload to neighbors or locally process it, under a given power constraint. A distributed optimization algorithm based on Alternating Direction Method of Multipliers (ADMM) via variable splitting is proposed. This allows to achieve the optimal workload allocation solution that maximizes QoS of users under the given power efficiency. In this work, power efficiency of each fog node is measured by the amount of consumed energy to offload each unit of workload from the cloud. Note that their work does not use the concept of VMs nor DDF programming model, and the considered environment is static (i.e., nodes and clients are static), avoiding the inherent migration problems.

### 2.3.4 Cost-Aware

As aforementioned, besides guarantying QoS to its users, fog service providers also need to maximize their profit. Hence, it is important to develop an accurate cost model in order to accept and implement fog computing. Besides, similarly to what cloud does, fog has to implement a pay-as-you-go cost model in order to provide services on-demand to its users, without under- or over-provisioning, and charging a fair price. To this end, the cost model needs to apply a communication model, an energy model, and a resource utilization model.

In this context, L. Gu et al. [37] state the importance of fog computing in medical cyber-physical systems as the number of users grows. They state that different infrastructure service providers may apply different charging policies. Therefore, in this paper, the authors aim to minimize the overall resource
management cost while satisfying the QoS requirements. They formulate the cost minimization problem in a form of Mixed-Integer NonLinear Programming (MINLP) with joint consideration of communication BS association, subcarrier allocation, computation BS association, VM deployment and task distribution. To tackle the high computational complexity of solving this problem, they linearize it into a Mixed-Integer Linear Programming (MILP) problem. This way they are able to solve the optimal programming model using solvers such as CPLEX and Gurobi. However, it is still time-consuming due to the existence of many integer variables. To this end, they further propose an LP-based two-phase heuristic algorithm. It is worth noting that this work explores placement of VMs in fog computing, whereas it does not tackle this problem in mobile environments, disregarding both user and server mobility, consequently not addressing the inherent migration problems. Moreover, the proposed method to verify if the QoS requirements are met neither considers the server state nor network state.

O. Skarlat et al. [25] start by describing a conceptual framework for resource provisioning and service placement in fog. They consider the concept of fog colonies (refer to Section 2.2.2) using a cooperative execution of IoT applications (DDF programming model). Based on this concept, their work formalizes an optimization problem that aims to adhere to the deadlines of deployment and execution time of applications and to maximize the utilization of existing resources in fog, rather than in cloud, leading to lower execution cost. To solve this placement problem, they apply different approaches, namely the exact optimization method and its approximation through a greedy first fit heuristic and a Genetic Algorithm (GA). They also compare the results in the fog simulation toolkit iFogSim to a classical approach that neglects fog resources and runs all services in a centralized cloud. The goal of the evaluation is to identify the best approach to solve the proposed optimization problem in terms of resulting QoS, QoS violations, and cost. The latter is composed only by the execution costs in cloud infrastructures, neglecting execution costs in fog nodes. Their work does not provide mobility mechanisms, not addressing the inherent migration problems. Moreover, all the communications within each fog colony need to be performed through the respective fog orchestration control, introducing new non-negligible communication latencies. Also, the proposed method to verify if the QoS requirements are met considers neither the server state nor network state.

The work performed by T. Bahreini et al. [38] also addresses multi-tier placement (DDF programming model). The authors formulate the Multi-Component Application Placement Problem (MCAPP). Their objective is to find a mapping between components and servers, such that the total placement cost is minimized. This cost is composed of four types of costs at each time slot: (1) the cost of running one component in a specific server, (2) cost of relocating one component from one server to another, (3) communication cost between one component and the user, and (4) communication between components. With the objective to minimize the overall cost incurred when running the application, they formulate the offline version of the problem as a Mixed Integer Linear Program (MILP) and then developed a heuristic algorithm for solving the online version of the problem. The algorithm is based on an iterative matching process followed by a locals search phase in which the solution quality is improved. This way they use simple algorithmic techniques, avoiding complex approaches such as those based on MDPs. They state that the proposed algorithm has low complexity and adds a negligible overhead to the
execution of the applications. Although this work considers the location of servers in the estimation of costs (2) through (4), it does not consider an environment with mobile fog nodes. Also, it only considers the presence of only one user with only one application. Moreover, in each time slot, each server is used by at most one component, not properly taking advantage of fog computing.

A different approach was taken by D. Ye et al. [39]. They leverage the characteristics of buses and propose a scalable fog computing paradigm with servicing offloading in bus networks. Knowing that buses have fixed mobility trajectories and strong periodicity, they consider a fog computing paradigm with service offloading in bus networks which is composed by roadside cloudlets and bus fog servers. The roadside cloudlet consists of three components: dedicated local servers, location-based service (LBS) providers, Access Points (APs). The dedicated local servers virtualize physical resources and act as a potential cloud computing site. LBS providers offer the real time location of each bus in bus networks. APs act as gateways for users and bus fog servers within the communication coverage to access the roadside cloudlet. As cloudlets have limited computational and storage resources, they may become overloaded. The bus fog server is a virtualized computing system on bus, which is similar to a light-weight cloudlet server. Hence, those buses not only provide fog computing services for the users on bus, but also are motivated to accomplish the computation tasks offloaded by roadside cloudlets. This allocation strategy is accomplished using GA, where the objective is to minimize the cost that roadside cloudlets spend to offload their computation tasks. Note that there is only considered mobility of fog nodes (i.e., users are static). Also, their problem assumes the applications are deployed as a whole, not addressing the DDF programming model advantages and difficulties in fog computing. Nonetheless, the proposed method to verify if the QoS requirements are met do not consider the servers state nor network state.

### 2.3.5 Multi-Objective

In some cases rather than only one objective, it might be crucial to improve the system performance from several perspectives/objectives. However, those can be independent and conflicting objectives. Therefore, unlike the previous sections, the current one aims to present works that were intended to study multi-objective migration optimization algorithms.

Y. Nan et al. [40] aim to provide an energy-efficient data offloading mechanism to ensure minimization of long-term system cost (measured by the money spending on energy consumption) and yet guarantee that users do not perceive a poor QoS. Their work assumes that fog nodes have two sources of energy. The primary source is the solar or green energy which has no monetary cost, however it is finite (in each time slot, the volume of electricity converted from solar energy is a stochastic value depending on the weather conditions). As a backup energy supply, fog nodes have also access to the non-free grid or brown power supply. In addiction, they also assume the presence of cloud data centers which, in this case, have only access to the grid power supply. Their work describes an online adaptive algorithm, Lyapunov Optimization on Time and Energy Cost (LOTEC). LOTEC is a quantified near optimal solution and is able to make control decision on application offloading by adjusting the two-way trade-off between
average response time and average cost. This decision-making distributes the incoming applications to the corresponding tiers without a priori knowledge of users and system status. Note that, in this work, there are no VM support nor DDF programming model. Also, there is no fog cooperation and both users and fog nodes are static, not addressing the inherent problems of migration.

The work performed by L. Yang et al. [41] aim to minimize both the average latency of all the users’ request loads and the overall costs of service providers. The latter is composed by minimization of both resource usage on cloudlets and service placement transitions. The authors state that this three-way trade-off is a difficult problem. Moreover, the request load could vary significantly and frequently in both spatial and temporal domain due to the mobility of users. Such dynamic request load implies a periodic update of decisions, keeping in mind both the current performance and the expected future workload (using user's mobility pattern and services access pattern to predict the distribution of user's future requests). In order to solve this three way trade-off, the authors first formulate the snapshot problem, named Basic Service Placement Problem (BSPP), which aims to optimize the access latency with the capacity constraints of cloudlets. As it is hard to solve, they design a competitive heuristic to BSPP which outperforms a set of benchmark algorithms. Their work further extends BSPP in order to minimize the above three-way trade-off. To do so, they normalize the costs and apply a weighted sum, allowing the formulation of a single objective problem. It is worth noting that this work does not consider DDF programming model, energy and bandwidth models, routing problems nor mobility of fog nodes.

L. Wang et al. [42] address the social VR applications to study the problem of placing VMs deployed in fog environments such that, the total cost in the overall cost in the fog system is minimized. Although motivated by VR applications, the authors state that this problem is fundamental for any applications that require interactions between either mobile user and the respective VM or user and VMs of other users. The placement problem is to decide where to place the service entity of each user among the cloudlets in order to achieve economical operations of cloudlets as well as QoS. This problem is non-trivial due to the following challenges: (1) cloudlets are heterogeneous in terms of activation and running costs, (2) VMs need to exchange metadata frequently with the associated users and other VMs (of other users), and (3) due to the fact that cloudlets are not intentionally designed to simultaneously accommodate many VMs, especially for VR applications where specific hardware such as GPU may be involved, resource contention needs to be controlled. The authors model the aforementioned challenges with four types of cost: activation cost, the placement cost, the proximity cost, and the collocation cost. These costs are then formulated as a single objective problem by using a weighted sum. They formulate the problem as a combinatorial optimization, which is NP-hard. To solve the problem, they propose iIterative Expansion Moves (ITEM) algorithm, a novel algorithm based on iteratively solving a series of minimum graph cuts. The algorithm is flexible and is applicable in both offline/static (i.e., no movement) and online/dynamic (i.e., with users mobility) scenarios. It is worth noting that this work does not consider fog nodes mobility.

Motivated by the trade-off between local execution power consumption and the offloading delay, the work performed by L. Liu et al. [43] has the objective to minimize energy consumption, delay, and payment cost (E&D&P) for mobile devices in fog computing environments, using queuing theory. Specif-
ically, three types of queues are applied, namely: mobile devices are considered as a M/M/1 queue, fog node as a M/M/c queue with a defined maximum request rate, and cloud as a M/M/∞ queue. Both wireless transmission and computing capabilities are explicitly and jointly considered when modeling this three-way trade-off. They formulate the optimization problem by finding the optimal offloading probability and transmit power. Using the scalarization method, they were able to transform the multi-objective into a single-objective optimization problem. In order to solve that single-objective problem they proposed an Interior Point Method (IPM)-based algorithm which can reduce the accumulated error and improve the calculation accuracy during the iteration process effectively. Note that, in the considered system, there is no movement and there exists only one fog node, which does not fulfill the ubiquity characteristic of fog computing.

L. Wang [44] et al. address two categories of costs, namely: static, which includes the operation cost and the service quality cost, and dynamic, comprising the reconfiguration cost and the migration cost. While the former is independently incurred inside each time slot, the latter is only charged for decision transitions across consecutive time slots. Operation cost refers to the incurred cost in terms of resources utilization (i.e., Central Processing Unit (CPU) and memory) or energy in each cloudlet. Service quality cost, which aims to capture the user perceived QoS, is proportional to the network delay between the user and its workload which may be distributed over several cloudlets. Reconfiguration cost regards to the increase of workload across time slots in each cloudlet. Finally, the migration cost includes both bandwidth cost on the network and the migration delay (both moving out of and into each cloudlet). The single objective problem formulation takes into account all these costs using a weighted sum. They propose Mobility-agnostic Online Edge Resource Allocation (MOERA) based on the “regularization” technique, which decomposes the problem into subproblems and solve them using convex programming. This algorithm receives as input the user’s workload and location and decides how resources should be allocated, such that the workload demands from every user is fulfilled while the overall cost system is minimized. Note that their work does not consider the DDF programming model, avoiding its difficulties and advantages in fog computing, and all fog nodes are static.

2.3.6 Concluding Remarks

The presented literature addresses different objectives regarding optimization in fog environments. For instance, in Section 2.3.1, the conferred works perform a single objective optimization in order to minimize the QoS offered to the end users. The works in Section 2.3.2, Section 2.3.3 and Section 2.3.4, also perform a single objective optimization, however, besides taking into consideration the QoS, they also consider bandwidth usage, energy and cost, respectively. In Section 2.3.5 were presented works that combine, in a multiple objective optimization, some of the above mentioned objectives. For the sake of analysis between the works described above, Table 2.2 presents a comparison of the features supported, and Table 2.3 compares the problem formulation, from whose perspective the problem is being optimized, as well as the algorithm(s) implemented in order to solve the problem formulation.

Although these approaches contribute to the improvement of fog computing, they do not account all
the aspects that this work aims to cover. Regarding Table 2.2 it is noticeable that there is lack of support for using DDF programming model. As already discussed, in Section 2.2.1, it can bring advantages to fog computing. Also, some works do not support the use of VM, considering, in this case, only the workload that needs to be processed. However, as previously mentioned, in Section 2.2, one of the main goals of fog computing is to support running multiple IoT applications at the same time. To do so, it is mandatory to use some kind of virtualization technique as discussed in Chapter 1. It also is clear that the reviewed works fail to consider dynamic environments with both client and fog mobile nodes. Even though some take into account mobility either from the mobile devices or fog nodes, none of them acknowledges both simultaneously. Migration is also a challenge little explored. However, due to the fact that fog is used in dynamic environments, it is crucial to support migration in order to rearrange the placement of applications or modules whenever needed. Finally, in order to improve the system flexibility, the presence of multiple fog nodes and the cooperation between them is also essential, however, this is not considered in some cases.

With respect to the optimization problem, Table 2.3, there are several approaches. These works aim to optimize their formulated problem from different perspectives. The fog provider perspective owns a fog infrastructure which may request resources from a cloud provider. The system provider assumes the ownership of both fog and cloud infrastructures. The fixed fog provider (special case of fog provider) perspective assumes only the possession of the fixed fog infrastructure. The service provider or broker, as the name suggests, aims to provide a service to its users, however it does not owns any physical infrastructure. Finally, the mobile devices perspective is similar to the service provider in the sense that it needs to request resources from fog and/or cloud providers, however, in this case, the objectives refer to the mobile devices (e.g., minimize the energy consumption of mobile devices).

Independently from the optimization perspective, there are some flaws in the reviewed literature. First, in order to support real-time IoT applications, its demands in terms of response deadline should be
Table 2.3: Problem comparison of the above described works.

<table>
<thead>
<tr>
<th>Ref.</th>
<th>Optimization perspective</th>
<th>Objectives</th>
<th>Variables</th>
<th>Constraints</th>
<th>Optimization manner</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>QoS cost</td>
<td>Bandwidth cost</td>
<td>Power cost</td>
<td>Operational cost</td>
<td>Placement</td>
</tr>
<tr>
<td>[31]</td>
<td>Fog provider</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[32]</td>
<td>Fog provider</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[33]</td>
<td>System provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>[34]</td>
<td>Fog provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>[35]</td>
<td>System provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[36]</td>
<td>Fog provider</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[37]</td>
<td>Service provider</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[38]</td>
<td>Service provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[39]</td>
<td>Fixed fog provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[40]</td>
<td>System provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[41]</td>
<td>Service provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[42]</td>
<td>Service provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[43]</td>
<td>Mobile devices</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[44]</td>
<td>Service provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a constraint. For instance, considering a hard real-time application (e.g., autonomous car controller), it is absolutely imperative that responses occur within the required deadline. In this regard, it is mandatory to take into account the servers and network states. However, as aforementioned, most of the works which define QoS constraints do not take these parameters into consideration. Note that the works performed in [35] and [40] were able to compute the QoS taking in consideration these parameters because there was no fog cooperation (i.e., the path is always client-fog node-cloud), and in each node the total amount of workload was known. Similarly the work performed in [33] was also capable to do it, however, as there was fog cooperation, the key element was to know each data and migration routing. Finally, the work in [34] also considers these parameters, however, in this case, those were captured with their refined hybrid push/probe technique which introduces some overhead to the system. From a similar perspective, users may also demand a maximum time to migrate its service due to service degradation during this period. As shown, in Table 2.3, none of the reviewed works has addressed this issue. Finally, some works do not consider the amount of resources each node can provide (i.e., CPU, memory and storage) and/or do not consider the amount of bandwidth available in the links. As aforementioned, in Section 2.2.1, nodes can be anything with computational and storage resources and the communications can also be of any type. This way, these constraints should be accomplished in order to ensure that the solution found does not exceed the available resources in each link and node.

As above mentioned, some flaws were identified in the reviewed literature. The aim of the current work is to propose a novel architecture which allows to overcome them. To do so, this architecture should be flexible enough to allow supporting different applications, each with an arbitrary number of modules with different demands encapsulated in VMs (i.e., allow the use of VMs and DDF programming model). These applications may be deployed by different users located at arbitrary locations. Servers in our ar-
chitecture should also be able to support multiple VMs at the same time (if there exists enough available resources), and be able to communicate between them (if there exists some connection), allowing the fog cooperation feature. Moreover, as discussed in Section 1.1, this work aims to implement fog computing in mobile environments, thus it is also objective to support mobility of both users and fog nodes, as well as location awareness. Meanwhile, as there is movement, connections will be changed as time passes (e.g., handovers may occur and bandwidth of mobile connections may change), therefore some rearrangements in the placement of VMs, through migration, may be necessary in order to ensure all deadlines are met. Nonetheless, as mentioned in Chapter 1, the presence of cloud servers is important to support fog computing, thus it is also an objective to include them. With the above mentioned features, the proposed architecture covers all the features presented in Table 2.2.

Finally, regarding the decision-making algorithm, the proposed architecture assumes the perspective of a system operator (i.e., owns both fog and cloud servers). In this perspective, the main goals are the following: to minimize the energy consumption of the nodes (by considering processing and communication operations) while keeping the percentage of processor and link resource usage as low as possible. This is important because, if a new client enters in the system and asks to deploy a new application and in the surrounding nodes there is no available resources, the system either needs to migrate some VMs in order to support the new client or it denies the access to the user. In either case the operator will have negative effects. This is also applied to the case where a deadline from a current running application is no longer ensured. In this case, the operator needs to migrate some VMs, however if the surrounding nodes have no more available resources, migrations might be longer or the number of migrations might be higher. These objectives must be minimized while ensuring the resources of nodes and links are not exceeded and the QoS deadlines, both during the application execution and migration, are met. As discussed above, in order to compute the QoS in the worst case scenario, both servers and network states should be considered by using, for that purpose, the variables placement, data and migration routing.

2.4 Modeling and Simulation Toolkits

As stated in Section 1.2, the proposed solution, which will be described later in Chapter 3, will be implemented in a suitable toolkit. In order to perform the toolkit selection, a survey was made on the currently available simulators. Table 2.4 compares fog and related computing paradigm simulators via comparison of their characteristics. The description of these characteristics are as follows:

- **Programming language.** Represents the programming language used to develop the simulation toolkit. This is important to evaluate the simulator simplicity, level of abstraction offered, maintainability, extensibility, etc.;

- **Documentation.** Denotes if there exists some kind of available documentation, which may include official documentation, tutorials, community, wiki, etc. This is of most importance once its absence may be an impediment to the extensibility and maintenance of the corresponding simulators;

- **Graphical support.** Depicts if the simulator provides a Graphical User Interface (GUI). This feature
is not crucial, however, it might be useful. Instead of defining the entire architecture programmatic-
ally, researchers can define it in a user-friendly environment;

- **Energy-aware.** Expresses if the simulator possesses awareness of the energy consumption of processing and communication operations. This feature is crucial in order to validate the system performance by allowing to verify the energy consumption of the nodes under different scenarios;

- **Cost-aware.** Similarly to energy-aware, cost-aware denotes it the simulator possesses awareness of the monetary costs involved in different operations (e.g., resource allocation, processing and communication operations);

- **Communication model.** Depicts if the simulator possesses and allows to implement communication mechanisms with different characteristics such as latency and bandwidth available. This feature is important to represent the diversity of communication types that fog computing supports;

- **Virtual machine support.** Shows if the simulator allows to encapsulate modules or applications within a VM. This is important for several reasons, as described in Chapter 1. Also, it allows to define applications or modules with different resource demands (e.g., CPU, memory and storage);

- **Application model.** Denotes if the simulator allows to define the application as a set of modules using the DDF programming model. This is an important feature from the fog perspective (refer to Section 2.2.1). Also, it allows users to define the required response deadlines upon the occurrence of a given event;

- **Migration support.** Expresses if the simulator possesses mechanisms to support the migration of applications and/or modules between different nodes. As shown in Section 2.3, this feature is important in order to rearrange the system according to the decision-making algorithm results;

- **Mobility/Location-aware.** Shows if the simulator supports the movement of clients and/or servers. As already mentioned, in Section 2.2, fog environments are dynamic by nature. Therefore, it is important to possess mechanisms allowing location awareness in order to benefit higher QoS;

- **Fog/Edge support.** Denotes if the simulator supports fog or edge computing. Since there are some major differences between these and cloud computing, as discussed in Section 2.1, this feature is of utmost importance.

Regarding Table 2.4, it can be clearly observed that most of the existing simulation toolkits are CloudSim-based. CloudSim is a well known simulator, in the cloud computing field, which has been used in several research studies [45]. More recent works in fog computing have begun to implement their investigation works in iFogSim [85], which is also based on CloudSim. Moreover, recently some other simulators have been proposed as extensions to iFogSim. Both CloudSim and iFogSim were developed by the CLOUloud computing and Distributed Systems (CLOUDS) laboratory [86], a software research and development group within the School of Computing and Information Systems at the University of Melbourne, Australia.

25
From Table 2.4, it is also evident that almost all simulators are Java-based, being that all opt for object-oriented programming. This brings the advantage of using a modular structure for programs which makes it good for defining abstract datatypes in which implementation details are hidden. Also, it helps in maintenance and extensibility. Since the design is modular, part of the system can be updated without a need to make large-scale changes. Moreover, it is also perceptible that documentation in many cases is not available. Note that empty cells represent little to no documentation. This is clearly a negative factor to corresponding simulators, specially for “isolated” simulators (i.e., those who are not based on CloudSim nor iFogSim). Looking through the graphical support column, in many cases there is no support. Although not a deciding factor, having this feature has a positive appreciation.

As aforementioned, in Section 2.3, energy is one of the objectives that this work intends to cover. When implementing the decision-making algorithm, the more realistic the energy model, the more realistic the algorithm will be. Looking through Table 2.4, it is perceptible that most of the simulators has some kind of energy awareness in what respects to the processing and communication models. However some have different approaches and granularity levels. For instance, CloudSim provides energy-conscious resource management techniques/policies. It supports modeling and simulation of different

<table>
<thead>
<tr>
<th>Simulation Platform</th>
<th>Programming language</th>
<th>Documentation</th>
<th>Graphical support</th>
<th>Energy-aware</th>
<th>Cost-aware</th>
<th>Communication model</th>
<th>Virtual machine support</th>
<th>Application model</th>
<th>Migration support</th>
<th>Mobility/Location-aware</th>
<th>Fog/Edge support</th>
<th>Last commit</th>
<th>Web page</th>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudSim</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2016</td>
<td>[45] [46]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudNetSim++</td>
<td>C++</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2015</td>
<td>[47] [48]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GreenCloud</td>
<td>C++/Otl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2016</td>
<td>[49] [50]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iCanCloud</td>
<td>C++</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2015</td>
<td>[51] [52]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudSched</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2015</td>
<td>[53] [54]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudAnalyst*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2009</td>
<td>[45] [55]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynamicCloudSim*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2017</td>
<td>[56] [57]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudReports*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2012</td>
<td>[58] [59]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealCloudSim*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2013</td>
<td>[60]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCSim</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2014</td>
<td>[61] [62]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudSim Plus*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2018</td>
<td>[63] [64]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudSim Automaton*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2018</td>
<td>[65]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISSECT-CF</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2018</td>
<td>[66] [67]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WorkflowSim*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2015</td>
<td>[68] [69]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud2Sim*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2016</td>
<td>[70] [71]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudSimDisk*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2015</td>
<td>[72] [73]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iFogSim*</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2016</td>
<td>[74] [75]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyiFogSim**</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2017</td>
<td>[76] [77]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iFogSimWithData Placement**</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2018</td>
<td>[78] [79]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EdgeCloudSim</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2018</td>
<td>[80] [81]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YAFS</td>
<td>Python</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2018</td>
<td>[82]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FogTorch</td>
<td>Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2016</td>
<td>[83] [84]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
power consumption models and power management techniques. In contrast, GreenCloud is a more fine-grained simulator to this end. Its energy models are implemented for every data center element. Moreover, due to the advantage in the simulation resolution, energy models can operate at the packet level as well. This allows updating the levels of energy consumption whenever a new packet leaves or arrives from the link, or whenever a new task execution is started or completed at the server. Nonetheless, this simulation resolution comes with a cost. In general, CloudSim, being an event-based simulator, is faster and scales to a larger number of data center nodes. A typical data center simulated in GreenCloud can be composed of thousands of nodes while the Java-based CloudSim can simulate millions of computers [50].

Cost minimization is not an objective from this work perspective, once it assumes, as stated in Section 2.3, the ownership of both fog and cloud server. Nonetheless, one might address the problem from other perspectives. For instance, from a broker perspective, which relies its resources on third party entities, this feature is important. It allows to validate the system performance by verifying its monetary costs under different scenarios and different decision-making algorithms. Therefore, in order to provide flexibility to the system, it is important to have some kind of cost awareness. Similarly to the energy-aware feature, cost awareness is covered by most of the simulators. For instance, ClouSim is able to keep tracking of the monetary costs related to resource usage such as memory, storage, bandwidth and task units (application service requests) that are served by the application services.

Regarding the communication between nodes, as stated above, it is also important to provide diversity due to the fact that fog networks rely in a wide range of communication technologies. Similarly to the energy awareness, some simulators have implemented different approaches and granularity levels. For instance, CloudSim can model network components, such as switches, but lacks fine-grained communication models of links and Network Interface Cards (NIC) causing VM migration and packet simulation to be network-unaware [48]. CloudNetSim++, on the other hand, supports a simulation model of real physical network characteristics such as network congestion, packet drops, bit error, and packet error rates. Moreover, GreenCloud allows communications based on TCP/IP protocol. It allows capturing the dynamics of widely used communication protocols such as IP, TCP, UDP, etc. Whenever a message needs to be transmitted between two simulated elements, it is fragmented into a number of packets bounded in size by network Maximum Transmission Unit (MTU). Then, while routed in the data center network, these packets become a subject to link errors or congestion-related losses in network switches [50]. Once again, these approaches with such high network resolution come with a cost with respect to the simulation complexity and execution time.

From Table 2.4, looking through the VM support and application model columns, it is noticeable that most simulators support this feature, however the migration support in fog or edge environments is not always supported. Moreover, the movement and location awareness is clearly one of the less explored features. For instance, in cloud environments, CloudAnalyst [55] is a tool whose goal is to support the evaluation of social network applications, according to the geographic distribution of users and data centers. Meanwhile, in fog environments, MyiFogSim explored users movement while supporting the migration of their applications based on a location awareness mechanism in order to benefit higher QoS.
On the one hand, this simulator approach introduces the migration policy which is responsible to define when the VM should be migrated using the user movement characteristics (i.e., position, speed, and direction) in order to guarantee QoS. On the other hand, upon the decision to migrate, it introduces the migration strategy which defines where and how the VM should be migrated. During the process, the latter regards to the type of migration (i.e. non-live container/VM migration or VM live migration using the post-copy method) and simple greedy strategies which define the fog node destination such as the one offering the shortest physical distance or lowest network latency. However, aside from its rough programming development, regarding the objectives of this work, it has also followed an oversimplified approach. For instance, this simulator throws some exceptions at the beginning of the simulation, not allowing to validate its performance. Also, the developed code is disorganized and it was not cleaned. Moreover, it does not use object oriented programming concepts such as inheritance or polymorphism.

Meanwhile, the implemented migration approach, as aforementioned, uses a greedy method which, as some reviewed works have shown (e.g., [32, 34]) do not allow to achieve a good system performance. Also, this method only considers the improvements in terms of distance or network latency, disregarding the impact that migrations cause on servers and network links which, in turn, impact other users perceived QoS. Furthermore, it does not use the network resources in order to migrate the VMs. Instead, it uses some delays and then, the VMs are migrated instantaneously between nodes. Hence, in this matter, no simulator has reliably explored node movement and how migrations of VMs can improve the QoS of the services.

Keeping in mind the above mentioned factors, the CloudSim “family” of simulators is the better fit for this work. This is motivated by the fact that these count with a larger community, information and documentation compared to other isolated simulators. Also, there is no need for such low level of network resolution. Although in CloudSim, and its extensions, built-in communication models are unrealistic by disregarding low-level network issues such as link errors or congestion-related losses, it allows to treat them as high-level attributes such as latency or bandwidth of connections. The same is applied to the energy awareness by allowing to modulate energy models according to the CPU usage of the nodes. Furthermore, as network usage is already measured, it is possible to use those values in the energy consumption calculation. Also, its cost awareness already varies according to the memory, storage, bandwidth and CPU utilization. Nonetheless, this is also motivated by the fact that it already has built-in features such as application model, virtual machine and migration support. As the only simulator developed by CLOUDS which already supports fog computing features is the iFogSim, this work will take this simulator as its basis. Although with some drawbacks, which will be explored in Chapter 3, such as not providing communication between fog nodes at the same hierarchical level, and assuming that both fog nodes and end devices are static, it has already several built-in features which meet the objectives of this work.
2.5 Summary

Regarding the surge of ubiquitous demand on high-complexity and quality of mobile services, fog computing is a promising computing paradigm. Motivated by the long and unpredictable end-to-end communication latency experienced in cloud computing and the explosive growth of mobile traffic, fog computing emerges as the most comprehensive and natural paradigm to support real-time applications and to get more efficient with the data sent to the cloud. Among other advantages, this idea is supported by its heterogeneity, ubiquity, scalability and real-time interactions support. Some research studies were already performed in this field in order to improve and validate the concepts of fog computing. From the analysis and discussion present in Section 2.3 it is noticeable the lack of research in dynamic environments with both client and fog mobile nodes, in using application partitioning through DDF programming model and in reliably ensuring QoS to its users. Likewise, it is also notorious that the proposed decision-making algorithms, in most of the cases, only consider few objectives and assume other objectives do not affect the problem. In order to modulate and simulate a fog (or related paradigm) computing system, there are several available choices, however none of which covers all the features that this work intends to address and study. Based on the analysis and discussion performed in Section 2.4, iFogSim was the chosen simulator as basis to test and validate the proposed architecture (presented in Chapter 3).
Chapter 3

Model Implementation

In this chapter, the fog computing components and the main design choices are presented and described. The main objective of fog computing, due to its heterogeneous nature, is to support delay sensitive applications, while being defined with any form of infrastructure. The latter can be structured by a network of static and mobile nodes with processing capacities interconnected using a wide range of communication technologies. In this way, in order to meet these requirements, the proposed architecture must be flexible enough to be able to support different nodes, including their mobility characteristics (regarding both its physical locations and migration of services), use different types of communication technologies, and to support different types of applications (each of which with different characteristics and QoS demands). On the other hand, the proposed solution must also be able to optimize the system performance in the presence of relevant events, while respecting the users defined QoS demands. As the system may not always be able ensure every defined deadline due to the hardware limitations of fog nodes, the aim is to maximize the number of applications which, in the worst case scenario, have their deadlines met. After an introductory overview of the architecture of the system (Section 3.1), Section 3.2 presents the assumptions adopted in order to deal with such huge problem and number of variables composing fog systems and its orchestration. This is followed by Section 3.3 which presents the proposed problem formulation. Finally, Section 3.4 presents the implementation details of the proposed solution which allows to implement different scenarios and evaluate the system performance while using different decision-making algorithms.

3.1 Architecture

The proposed architecture comprises entities and services. The physical topology follows the typical architecture of fog computing presented in Section 2.2. Therefore, it is composed of entities such as client, fog and cloud nodes. Additionally, depending on the type of services, clients may also be attached to entities such as sensors and actuators. As present in [11], fog computing relies in an infrastructure. Thus, there is always a central, static and interconnected network of fog and cloud nodes. Meanwhile, at the edge of the network there might exist some mobile fog nodes deployed inside buses, trains, cars,
etc. Similarly to the fog nodes, clients may also be static or mobile. Regarding the communication links, static nodes might be connected to an arbitrary number of nodes using different types of connections. Meanwhile, mobile nodes are always connected to their closest static fog or cloud node via cellular network. In this architecture, clients deploy an arbitrary number of applications/services, each of which structured as a collection of modules using DDF programming model (refer to Section 2.2.1). Moreover, the presence of a central control unit fully aware of the system performance and its characteristics is also assumed. The full awareness of the system encompasses the knowledge of every application/service, sensor, actuator, node and link characteristics. Based on this information, the controller looks into the network as a directed graph where vertices represent nodes and edges represent network links. Note that nodes are not directly connected to every other node, however there is always at least one network path between any pair of nodes. Also, it looks into each application as a Directed Acyclic Graph (DAG) where vertices represent modules and edges represent data dependencies between modules (refer to Section 3.1.2). With all the information required, the controller runs the decision-making algorithm and, depending on the solution found, orders nodes to perform the decided actions. Each node, according to the central control unit instructions, is then responsible for hosting modules and processing the respective data.

3.1.1 Entities

As above mentioned, the physical topology is composed of sensors, actuators, and client, fog and cloud nodes. This section presents a brief description of each entity and how the interaction between them is performed.

Sensors

Sensors are entities collocated at the client nodes. Each sensor is defined by attributes ranging from its connectivity to output attributes. Regarding the connectivity, each sensor is defined with a given node gateway (i.e., the respective client node) to which the sensor is connected. Meanwhile, the output attributes define the data chunk (i.e., tuple) type to be processed (described below) it generates at its output, the destination module, and also the distribution of tuple inter-transmission or inter-arrival time - which identifies the tuple arrival rate at the gateway. This tuple arrival rate can assume a deterministic, normal or uniform distribution. In order to perform the tuple arrival rate distribution, once the sensor is started or whenever it sends a tuple to its gateway, it schedules another event with a delay calculated based on its distribution, upon which it generates and sends another tuple, and so on.

Actuators

Similarly to the sensors, actuators are entities collocated at the client nodes. Also, each actuator is associated with a gateway to which it is connected. Actuators represent the final step in the sense-process-actuate model, as present in Section 2.2. Typically, it represents the final step of an application loop (refer to Section 3.1.2). Therefore, upon the reception of a given tuple (event), it can be measured
the total loop delay (i.e., end-to-end delay) allowing to verify the QoS offered to the corresponding application.

Tuples

Tuples form the fundamental unit of communication between entities in the system. Whenever a given system entity (i.e., a node or a sensor) needs to send a piece of data to another entity (i.e., node or an actuator), it generates a tuple and sends it to either its gateway or the next node given by its data dependency routing table. By means of simplicity, as stated in Section 2.4, tuples are not fragmented, instead these are sent as a single packet through the corresponding network links. A tuple is characterized by the identifiers of its source and destination modules, as well as its type. It has also attributes specifying its resource requirements in terms of processing, defined as Million Instructions (MI), and network, defined as the length of the encapsulated data (in Bytes). Additionally, whenever needed, a special type of tuples is used to encapsulate all the data required to migrate an application model between entities.

Nodes

Although this work considers three different types of physical nodes (i.e., cloud, fog and client nodes), their characteristics are identical. The main differences lie, in their hardware characteristics, i.e., in the quantity of available resources and in its power consumption. Node resources consist of processing capacity (defined as Million Instructions Per Second (MIPS)), memory (in Bytes), and storage capacity (in Bytes). Regarding the power consumption, there exist two levels. On the one hand, the static, idle or leakage power consumption, is mainly caused by the leakage currents and is unrelated to the usage of computing resources. On the other hand, the dynamic or busy power is the result of circuit activity and is the result of the usage of computing resources. Meanwhile, the power consumption for wired communications is considered to be zero and for mobile communications is a constant arbitrary value. Nonetheless, nodes are also characterized by resource utilization prices. These include the price of processor, memory, storage and bandwidth usage, as well as the price per unit of energy consumption.

During the simulation, nodes are responsible for executing some actions upon the occurrence of events. These are listed and described below. Note that the event name is followed by its data carried in parentheses.

- **LAUNCH_MODULE (application module)**. This event is scheduled by the controller and it only occurs at the beginning of the simulation. It carries the application module which the node is responsible for hosting. Upon its occurrence, the node is responsible to allocate the required resources, to encapsulate application module within a VM, and to initialize its periodic application edges, if any (i.e., periodic data dependencies between modules - refer to Section 3.1.2). This is performed by scheduling, for each periodic edge, an event defining the time instant when the corresponding tuple should be sent;

- **SEND_PERIODIC_TUPLE (application edge)**. When this event occurs, the node generates a tuple (corresponding to the received application edge) and, based on its data dependency routing
table, send it to the correct network link tuple queue (described below). Then, it is also responsible to schedule the next event defining the time instant, based on the edge periodicity, when this event should occur again;

- **TUPLE ARRIVAL (tuple).** As stated above, tuples may encapsulate an application module or a data dependency. On the one hand, if the node receives a tuple containing an application module within a VM which is directed to itself, it allocates the required resources and starts to perform its setup. This is performed by scheduling an event which defines when the VM setup has finished and it is ready to execute. Otherwise, it resorts to its migration routing table in order to submit the tuple to the right network link tuple queue. On the other hand, if the node receives a tuple containing a data dependency meant for one of the modules it hosts, it submits the tuple for processing. Otherwise, it uses its data dependency routing table in order to submit the tuple to the right network link tuple queue. Nonetheless, there might occur the case in which, upon the reception of given tuple, the node does not know what action to perform. This occurs whenever a tuple is meant to be delivered to an application module which is currently in migration or setup process or some handover has occurred and the data dependency routing table has been changed. In either case, the tuple is dropped/ignored and, in order to evaluate the QoS, it is incremented the number of tuples lost;

- **UPDATE TUPLE QUEUE (destination id).** Network links between nodes only allow to transmit a tuple at a time. Therefore a tuple queue is used to store the tuples which need to transmitted (one per link). In this work it is considered a First In, First Out (FIFO) queue. Note that it can be replaced with a priority-based queue as one might need. Whenever a tuple is send through a given network link, this event is scheduled based on the size of data that needs to be sent and the link state. Therefore, upon the transmission of a tuple, this event occurs allowing to send the next tuple in the queue. When the tuple queue is finally empty, the resource management operation is performed (described below);

- **RESOURCE MANAGEMENT ().** This event occurs whenever some resource usage has changed, e.g., when a tuple has started/finished to be processed, a network link starts to be used or is not used anymore, a migration starts (i.e., resources are freed), etc. Additionally, it is also a periodic event scheduled by the node itself in order to cover the cases where its state does not changes for a long period of time. When this event occurs, the node computes its own energy consumption and the total monetary price of resource usage according to its predefined prices. The energy consumption, is computed by the total power consumption (both in processing operations and transmission of data over the cellular network) and the time interval since the last event. In order to compute the processing power consumption at node $n$ based on its percentage of CPU usage, $c_n(t)$, at time $t$, this work assumes the linear power consumption model presented in Equation 3.1 [75], where $f_n^{iPw}$ corresponds to the leakage power consumption and $f_n^{hPw}$ to the dynamic power consumption at node $n$.

$$f_n^{Pw}(t) = f_n^{iPw} + (f_n^{hPw} - f_n^{iPw}) \times c_n(t).$$ (3.1)
Regarding the monetary cost, it is computed by the percentage of resource usage during the time interval since the last event. This computation is independent for each resource. For instance, the monetary cost for utilizing processing resources at node $n$ during the time interval $\Delta t$ is presented in Equation 3.2. Where $fP^{\text{Mips}}_n$ corresponds to the total monetary price per unit time of the processing resources at node $n$;

$$c^{\text{Mips}}_n(\Delta t) = fP^{\text{Mips}}_n \times c_n(t) \times \Delta t. \tag{3.2}$$

- **UPDATE_MOVEMENT ()**. This is also a periodic event scheduled by the node itself and, as the name suggests, it allows to update the movement of the (mobile) node. In this event, the current position (in a two-dimensional plane) is computed based on the previous position, direction (characterized by one of the eighth cardinal and ordinal directions), velocity and the time passed since the occurrence of the previous event of this type. Then, both velocity and direction are arbitrarily updated. Note that different nodes may follow different movement models;

- **REMOVE_CONNECTION (destination id)**. The central control unit is responsible to maintain each mobile node connected to its closest static fog or cloud node. To do so, it is responsible for monitoring the current position of each node and, whenever needed, perform the required handovers. Whenever an handover occurs, the controller creates the new connections between the involved nodes, and orders (in the form of events) the removal of the required connections. Therefore, whenever the node receives this event, it removes the connection to the indicated node;

- **MIGRATION (application module id, destination id)**. Similarly to connection removal, the central control unit, based on the solution found by the decision-making algorithm, may also ask to migrate a given application module to another node. In this case, the node is responsible for stopping the execution of the corresponding VM, freeing all resources reserved to support the execution of that module, and submit it, based on its migration routing table, to the corresponding network link tuple queue;

- **CHECK_PROCESSING_COMPLETION ()**. This event is scheduled whenever a tuple starts to be processed. The delay is computed based on its processing requirements (MI) and the node processing capacity (MIPS); Similarly to the network links, the CPU has also a FIFO tuple processing queue, thus it only processes one tuple at a time. Therefore, after processing a tuple, this event occurs allowing to process the next one in the queue. When the tuple queue is finally empty, it is performed the resource management operation;

- **FINISH_SETUP_MIGRATION (application module)**. This event occurs whenever the VM finishes its setup. It means the module is now able to run. Thus, similarly to what happens in the launch module event, the node is responsible to initialize its periodic edges (if any).

Nonetheless, clients form a special type of nodes. Although sharing all the previous characteristics and responsibilities, these, as stated above, may be attached to sensors and actuators. Thus, these are also responsible for receiving tuples from sensors or forwarding tuples to the actuators.
Note that inside each node there exist two routing tables. On the one hand, the data dependency routing table is composed of a mapping between a source and destination module (i.e., data dependency) and the next hop identifier. On the other hand, the migration routing table maps the application module to the next hop identifier.

Controller

As aforementioned, this architecture assumes the presence of a central control unit fully aware of the system performance and its characteristics. Based on the latter, it is capable of running the decision-making algorithm and, depending on the solution found, order nodes to perform the decided actions. With all the information required, before starting the simulation execution, the controller creates and runs the defined optimization algorithm (discussed in Section 3.2). Regarding the simulation itself, similarly to the above presented entities, it is also responsible for executing some actions upon the occurrence of events. The controller events are the following:

- **START_ENTITY ().** This event only occurs at the beginning of the simulation. Upon its occurrence, based on the solution found by the decision-making algorithm, the controller performs the placement of every module in the corresponding nodes, and defines all routing tables. It is also responsible to schedule the event (with an arbitrary time) used to stop the simulation;

- **UPDATE_TOPOLOGY ().** This is a periodic event scheduled by the controller itself. Upon its occurrence, it verifies, based on the current position of each node, if some handover needs to be performed in order to ensure that each mobile node is connected to its closest static fog or cloud node. Then, for each handover, the controller creates the new connection between the nodes and orders, under the form of events, the required nodes to remove the corresponding old connections. If some handover has been performed, the controller recomputes the decision-making algorithm. Then, based on the solution found, it is also responsible to update all data dependency and migration routing tables. Moreover, if new placements for one or more application modules were determined, it orders, under the form of events, the required nodes to start performing the required migrations;

- **UPDATE_VM_POSITION (application module id, destination id).** In order to let the controller know the current position of each application module at any time, before start sending a tuple containing a VM to another node, nodes inform, under the form of events, the controller to which node the VM is being forwarded. Therefore, upon the occurrence of this event, the controller updates in its algorithm information, the node in which the application module is located at;

- **STOP_SIMULATION ().** Upon the occurrence of this event, the controller stops the simulation and presents the simulation results. In this way, it is possible to evaluate the system performance under a given scenario and a given decision-making algorithm.
3.1.2 Services

As aforementioned, clients can deploy an arbitrary number of applications/services, each of which composed of several modules using DDF programming model modeled as DAGs. Each DAG is composed of vertices representing modules that perform processing on incoming tuples and edges denoting data dependencies between modules. The application components are listed and described below. Figure 3.1 presents the DAG of an example application.

An application module represents processing elements of the applications. It is a VM deployed within a given node. It is characterized by its resource demands in terms of processing, memory, storage and bandwidth. The application modules process each incoming tuple and generate output tuples which are sent to next modules in the DAG. The number of output tuples per input tuple is decided using a selectivity model - which can be based on a fractional selectivity or a bursty model (described below). Additionally, modules can assume one of the following types: client, global or normal (i.e., none of the previous). An application module of type client, needs to be processed within a client node (e.g., a graphical interface module). The global module type means that it is used by several clients. For instance, a Virtual Reality (VR) game provider may deploy a module which is used to compute the game global state. Therefore, this module cannot be hosted within a client node. Finally, type normal means that the module can be deployed within any fog or cloud node or even within the respective client node. Finally, modules can also be defined with a given deadline which defines the maximum allowed migration time.

An application edge denotes the data-dependency between a pair of application modules and represents a directed edge in the application model. Each edge is characterized by the type of tuple it carries, along with the processing requirements and length of data encapsulated in these tuples. An application edge can be defined as periodic or event-based. Tuples on a periodic application edge are issued at regular intervals, while a tuple on an event-based edge $e = (u, v)$ is sent when the source module $u$ receives a tuple and the selectivity model of $u$ allows the emission of tuples carried by $e$. For instance, regarding the Figure 3.1, the periodic application edge connecting $sensor$ to $u$, generates a given tuple every 10ms, which needs to be processed in module $u$. Meanwhile, the event-based application edge connecting $u$ to $v$ only issues a tuple if, upon the processing of a tuple in module $u$, the corresponding selectivity model allows to. As stated above, there are two types of selectivity model. On the one hand, the fractional selectivity generates an output tuple for an incoming input tuple with a fixed probability. On the other hand, the bursty model has two periods. During the high burst period, all input tuples result in an output tuple, while during the low burst period, no input tuples result in an output tuple.

Finally, application loops are used to specify the process-control loops of interest to the user. In this way, users may specify a given path within the DAG (i.e., a loop) by defining the list of modules starting from the origin of the loop to the module where the loop terminates, in order to measure its end-to-end latency. For instance, regarding the Figure 3.1, the application loop $i$ measures the total latency between the generation of a given tuple from the sensor until the corresponding tuple arrives in the actuator. Additionally, loops can also be defined with a given deadline, which defines the maximum allowed loop execution time.
3.2 Assumptions

In order to deal with such huge problem and number of variables composing fog systems and its orchestration, the following assumptions were adopted:

I There is always a connected infrastructure. As shown in Section 2.2 and presented in [11], fog computing relies in an infrastructure. Therefore, although this work supports mobility of fog and client nodes, there is always the need of a central, static and interconnected network of fog nodes. Additionally, there can exist some static cloud and client nodes. Note, that this is not a assumption, however it is still listed in order to emphasize the idea that fog computing is not meant to be constituted only by mobile devices. In such cases, there are more suitable computing paradigms such as MCC and MACC as stated in Section 2.1. This ensures there is always at least one network path between any two nodes. In order to accomplish this statement, and to avoid isolated node partitions, mobile nodes are always connected to their closest static fog or cloud node;

II Mobile nodes move in a 2D plane. This work assumes a 2D plane in which mobile nodes can move. Therefore, its position is defined with two axis. Moreover, the velocity is defined of its module and the direction of one of the eighth cardinal and ordinal directions (i.e., N, NE, E, etc.) at a time;

III The 2D plane has full cellular network coverage. Mobile communications are outside the scope of this work. Therefore, this work only considers the presence of a simplified full-duplex cellular Long-Term Evolution (LTE) network with full coverage in the whole 2D plane landscape. Moreover, a constant low latency and constant communication speed are assumed in the whole 2D plane landscape. These simplifying assumptions are meant to keep the focus of the model on fog computing infrastructure performance, eliminating parasite variables related with the impact of wireless communication performance;

IV Handovers are performed based the on RSS with threshold in a soft fashion. As stated before, mobile nodes are always connected to its closest static fog or cloud node. It is assumed that the distance is the only variable in the calculation of the RSS. Therefore, being connected to the closest node, means to be connected to the node that offers the best RSS. However, in order to prevent large number of handovers in border areas, also known as ping-pong effect, a threshold is
used, as depicted in Figure 3.2. It is clear that the mobile node's RSS from BS2 exceeds RSS from BS1 at point A, however the handover request is issued at point B, where the RSS from BS1 is lower than the threshold value and the RSS from BS2. Moreover, whenever an handover needs to be performed, it is executed in a soft fashion. This means that, the connection to the new channel is made before the connection from the source channel is disconnected. This is performed through the parallel use of source and destination channels over a period of time. In practice, this means that if an handover is issued while a given tuple is currently being transmitted, the tuple is actually sent, and only then the connection from the source channel is disconnected;

V Communication latency and bandwidth are used to mask network details. As already stated in Section 2.4, communication latency and bandwidth are used to overcome network characteristics such as packet losses, as well as to mask multi-hop communications. With the emergence of communication technologies such as optical fiber, the main delays are caused by the machines rather than the wires themselves. Therefore, instead of defining every single network component (e.g., routers, switches or APs) and the delay they would cause, the effects of the latter are abstracted in those two variables;

VI Bandwidth utilization is performed through time-division multiplexing. Similarly to what happens in [75], each network link only transmits one tuple at a time. Therefore, a FIFO tuple queue is implemented in each link in order to store all the tuples aiming to transmitted when it is still in use;

VII CPU utilization is performed through time-division multiplexing. Similarly to the network links, each CPU only processes one tuple at a time;

VIII Tuples are dropped when handovers or migrations are performed. As already stated in Section 3.1, whenever a node receives a tuple and it does not know what action to perform, the tuple is dropped/ignored. On the one hand, this case might occur during a migration, either because the module is still being transmitted over the network or because it is in the setup phase (it is considered an arbitrary delay to perform the VM setup upon its migration). On the other hand, it might occur after an handover is executed. For instance, consider an example with two static nodes, $A$ and $B$, connected using a wired link. Client $C$ (a mobile node), is connected via wireless connection to node $A$. The client has an application with two modules $M_1$ and $M_2$. The module $M_1$ is deployed in $C$ and $M_2$ in $B$. In this case, when module $M_1$ needs to send some tuple to
\( M_2 \), the tuple follows the path \( C \rightarrow A \rightarrow B \). At some point, an handover occurs and the client \( C \) becomes connected to node \( B \). At this point, when module \( M_1 \) needs to send some tuple to \( M_2 \), the tuple follows the path \( C \rightarrow B \). If in the meantime, some tuple was being transmitted from \( C \) to \( A \), as all routing tables were updated, when \( A \) receives the tuple, it no longer knows what action to perform. Nonetheless the number of dropped tuples is taken into account as a QoS metric;

IX  **Migration is performed using full VM migration.** When a migration needs to be performed, the application module stops executing, the VM is migrated to the destination node, and after the setup phase, the VM continues executing;

X  **Problem is addressed from the system provider perspective.** Although bearing in mind a flexible solution for other perspectives, as it will be discussed in Section 3.3, this work has addressed the problem from the system provider perspective. This means the proposed architecture assumes the ownership of all fog and cloud nodes;

XI  **The model assumes initial deployment done at \( t=0 \).** As discussed in Section 3.1.1, by the time the \texttt{START\_ENTITY} event occurs, the controller already has created and executed the decision-making algorithm in order to compute the optimal locations for each application module, based on which it deploys immediately the modules onto the corresponding nodes and updates their routing tables. This approach is motivated by the work performed in [75], which also assumes that all modules are already deployed at time \( t = 0s \) by assuming a null deployment time. Therefore, the model focuses on the steady state operation of the system;

XII  **Request patterns are known.** As already discussed in Section 2.2, all request patterns are known by the central control unit, allowing it to know the required resources for each module. For instance, sensors generate tuples following a distribution of tuple inter-transmission or inter-arrival time - which identifies the tuple arrival rate at the gateway. This tuple arrival rate can assume a deterministic, normal or uniform distribution. Moreover, each application edge can be defined as periodic or event-based, where the number of output tuples per input tuple is decided using a selectivity model - the latter can be based on a fractional selectivity or a bursty model. In reality this might not be always the case, however, in such cases, clients should be responsible to indicate the required resources;

XIII  **The problem size is constant along the simulation.** This work, motivated by the work in [75], also assumes a constant problem size, meaning that no clients enter or exit from the system and applications are meant to run during the whole simulation. Some works, such as the one presented in [25] consider one time request applications (application execute one request and exit from the system). However, this work considers applications that are deployed into the system and meant to execute multiple requests (e.g., the drone example presented in Chapter 1). If this assumption was not held, the decision-making algorithm would have to be executed whenever some of this events occurred;
XIV **Mobile nodes advertise the amount of local resources that they are willing to use.** Mobile devices are characterised for having battery constraints. Although in the case of mobile fog nodes, this might not be a constraint while being deployed inside buses, trains, cars, etc., in the case of mobile clients (e.g., mobile devices, drones, etc.), it might be. As there are no available mechanisms to represent the remaining available battery in each mobile device, in order to overcome this problem, two solutions were considered. On the one hand, it would be possible to implement a weight to represent the willing to waste energy or to use processing resources. In this case, static nodes, being connected to the power grid, would have this weight set to one, while mobile nodes would arbitrarily define the weight to an higher value. On the other hand, instead of allowing to use all its available resources, mobile nodes could advertise to the controller the quantity of processing resources which were willing to use. As both approaches would mask the problem in the same way, the choice fell on the second one;

XV **Clients can only process their own application modules.** As it will be discussed in Section 3.3, this work considers that clients can only process their own application modules. Nonetheless, the presented solution is flexible enough to allow the system provider, based on a given business model, to benefit from the processing capacity of client nodes to process application modules from other clients in order to further enhance the overall offered QoS as one might aim;

XVI **The execution of the decision-making algorithm is event-driven.** The controller only runs the optimization algorithm whenever a relevant event occurs, such as an handover. This is motivated by the work performed in [34], which states that running the decision-making algorithm periodically with a short recalculation interval introduces a considerable overhead to the server while, running with a long recalculation interval may introduce periods of transgression of QoS guarantees. In truth, there is no optimal time recalculation interval because it depends on many factors. Moreover, as the average values in the system remain constant, there is no need to recompute the algorithm until some of these event occur. Note that the decision-making algorithm is recalculated whenever an handover is performed because there might exist some paths which are no longer available;

XVII **Nodes can only use a part of their resources.** Definitely, in practice, there exist some overheads which cannot be ignored. These are mainly caused by operational maintenance which includes both transmission of control messages and migration of VMs between nodes. Therefore, it assumed that nodes can only use part of their resources (i.e., processor, memory, storage and bandwidth) to perform useful work.

### 3.3 Problem Formulation

Regardless of the topology/environment in question, the main goal of the system provider is to ensure applications QoS while optimizing its own objectives. This section presents a problem formulation, based on the gathered central control unit information, which will be solved using different optimization algorithms (see Section 3.4).
### 3.3.1 System and Problem Variables

This section presents a description of each system variable related to the entities and services that make part of the proposed architecture and, if applicable, how these are computed. Also, it presents the problem variables used to optimize the system performance. As discussed in Section 2.3.6, these are the placement, $P$, (representing the mapping between the application modules and the corresponding hosting nodes), the data dependency routing, $R$, (representing the routing path for each data dependency between pair of modules) and VM migration routing, $V$, (representing the routing path for each VM which needs to be migrated between nodes). For the sake of completeness, Tables 3.1 and 3.2 present the notation used for the problem formulation, the corresponding sizes, units and descriptions.

The physical topology is composed of $N$ interconnected nodes with $E$ edges. Each node, $n$, is characterized by its processing, memory and storage capacities, $f_n^{\text{Mips}}$, $f_n^{\text{Mem}}$, and $f_n^{\text{Strg}}$, respectively. Also, each node, $n$, is characterized by its idle, busy and mobile network transmission power consumptions, $f_n^{\text{IPw}}$, $f_n^{\text{Bw}}$, and $f_n^{\text{Tx}}$ respectively. Nonetheless, each node, $n$, may define its own prices for resource utilization of processing, memory, storage and bandwidth, and power consumption, $p_n^{\text{Mips}}$, $p_n^{\text{Mem}}$, $p_n^{\text{Strg}}$, $p_n^{\text{Bw}}$, and $p_n^{\text{Pw}}$ respectively. Additionally, once the system is being optimized from the system provider perspective, there are some differences between client and fog or cloud nodes (discussed below). Therefore, in order to differentiate them, $f_n^{\text{Fog}}$ was introduced to represent whether $n$ is a client node or not ($f_n^{\text{Fog}} = 0$ means node $n$ is a client, otherwise, $f_n^{\text{Fog}} = 1$). Still, regarding the physical topology, each edge, $e$, is characterized of its latency, available bandwidth, source and destination nodes, $E_e^L$, and $E_e^B$, and $E_e^S$, and $E_e^D$, respectively. Note that every two connected nodes have two directional links representing a full-duplex communication.

Regarding the service topology, each client may deploy an arbitrary number of applications each constituted by an arbitrary number application modules. In total, the system has $A$ applications and $M$ modules, which need to be hosted and processed. Each module, $m$, has its own resource requirements in terms of processing, memory and storage and a migration deadline, $m_n^{\text{Mips}}$, $m_n^{\text{Mem}}$, $m_n^{\text{Strg}}$, and $m_n^{\text{MigD}}$, respectively. While the memory, storage and a migration deadline are known variables, the amount of processing capacity is computed through the analysis of the application edges. In total, in the system there are $K$ application edges. As aforementioned, each application edge, $k$, is characterized by its resource requirements in terms of processing and bandwidth, $e_k^{\text{Cpu}}$ and $e_k^{\text{Nw}}$, respectively. Nonetheless, as stated in Section 3.1, application edges can be periodic or event-based, however, in either case these have a period and a probability of occurring which depends on the source and the path travelled inside the application DAG. For instance, regarding Figure 3.3, data dependency $e_1$ is characterized by a probability of 1 every 10 ms, while data dependency $e_2$ is characterized by a probability of 0.9 every 10 ms, and so on. Table 3.3 presents the application edges’ characteristics for the example application presented in Figure 3.3. Therefore, each application edge, $k$, is also defined by its periodicity, probability, source and destination modules, $e_k^{\text{Per}}$, $e_k^{\text{Prob}}$, $e_k^{\text{S}}$, and $e_k^{\text{D}}$, respectively. Note that even though, in this case, it is considered a deterministic distribution in the tuple rate of the sensors, it can be easily adapted to normal or uniform distributions. Also, in this case, event-based edges feature fractional selectivity, though they could also be adapted for bursty selectivity.
Table 3.1: System variables.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Unit</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
<td>1 × 1</td>
<td>Number of nodes</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>1 × 1</td>
<td>Number of applications</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>1 × 1</td>
<td>Number of network links</td>
</tr>
<tr>
<td>M</td>
<td></td>
<td>1 × 1</td>
<td>Number of modules</td>
</tr>
<tr>
<td>K</td>
<td></td>
<td>1 × 1</td>
<td>Number of dependencies</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>1 × 1</td>
<td>Number of module pairs with dependencies</td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td>1 × 1</td>
<td>Number of loop deadlines</td>
</tr>
<tr>
<td>f_{Fog}</td>
<td></td>
<td>N × 1</td>
<td>Indicates whether each node is a fog or client node</td>
</tr>
<tr>
<td>f_{Mips}</td>
<td>MIPS</td>
<td>N × 1</td>
<td>Processing capacity of each node</td>
</tr>
<tr>
<td>f_{Mem}</td>
<td>Byte</td>
<td>N × 1</td>
<td>Memory capacity of each node</td>
</tr>
<tr>
<td>f_{Strg}</td>
<td>Byte</td>
<td>N × 1</td>
<td>Storage capacity of each node</td>
</tr>
<tr>
<td>f_{Pw}</td>
<td>W</td>
<td>N × 1</td>
<td>Busy power consumption of each node</td>
</tr>
<tr>
<td>f_{Pw}</td>
<td>W</td>
<td>N × 1</td>
<td>Idle power consumption of each node</td>
</tr>
<tr>
<td>p_{Mips}</td>
<td>ε</td>
<td>1 × N</td>
<td>Price of using processing resources in each node</td>
</tr>
<tr>
<td>p_{Mem}</td>
<td>ε</td>
<td>1 × N</td>
<td>Price of using memory resources in each node</td>
</tr>
<tr>
<td>p_{Strg}</td>
<td>ε</td>
<td>1 × N</td>
<td>Price of using storage resources in each node</td>
</tr>
<tr>
<td>p_{Bw}</td>
<td>ε</td>
<td>1 × N</td>
<td>Price of using network resources in each node</td>
</tr>
<tr>
<td>p_{Pw}</td>
<td>ε</td>
<td>1 × N</td>
<td>Price of power consumption in each node</td>
</tr>
<tr>
<td>E^S</td>
<td></td>
<td>E × 1</td>
<td>Source node for each network link</td>
</tr>
<tr>
<td>E^D</td>
<td></td>
<td>E × 1</td>
<td>Destination node for each network link</td>
</tr>
<tr>
<td>E^L</td>
<td></td>
<td>s × E</td>
<td>Link latency for each network link</td>
</tr>
<tr>
<td>E^Bw</td>
<td>Byte/s</td>
<td>E × 1</td>
<td>Link bandwidth for each network link</td>
</tr>
<tr>
<td>m_{Mips}</td>
<td>MIPS</td>
<td>M × 1</td>
<td>Processing resources needed for each module</td>
</tr>
<tr>
<td>m_{Mem}</td>
<td>Byte</td>
<td>M × 1</td>
<td>Memory resources needed for each module</td>
</tr>
<tr>
<td>m_{Strg}</td>
<td>Byte</td>
<td>M × 1</td>
<td>Storage resources needed for each module</td>
</tr>
<tr>
<td>m_{MigD}</td>
<td>s</td>
<td>M × 1</td>
<td>Migration deadline for each module</td>
</tr>
<tr>
<td>e_{CPU}</td>
<td>MI</td>
<td>K × 1</td>
<td>Tuple CPU size needed to be processed for each dependency</td>
</tr>
<tr>
<td>e_{Network}</td>
<td>Byte</td>
<td>K × 1</td>
<td>Tuple network size needed to be sent for each dependency</td>
</tr>
<tr>
<td>e_{Pc}</td>
<td>s</td>
<td>K × 1</td>
<td>Periodicity of sending the tuple for each dependency</td>
</tr>
<tr>
<td>e_{Prob}</td>
<td>K × 1</td>
<td>Probability of sending the tuple for each dependency</td>
<td></td>
</tr>
<tr>
<td>e_{Source}</td>
<td>K × 1</td>
<td>Source module for each dependency</td>
<td></td>
</tr>
<tr>
<td>e_{Dest}</td>
<td>K × 1</td>
<td>Destination module for each dependency</td>
<td></td>
</tr>
<tr>
<td>l_{S}</td>
<td></td>
<td>Z × 1</td>
<td>Source module for each pair of modules with dependencies</td>
</tr>
<tr>
<td>l_{D}</td>
<td></td>
<td>Z × 1</td>
<td>Destination module for each pair of modules with dependencies</td>
</tr>
<tr>
<td>m_{Bw}</td>
<td>Byte/s</td>
<td>M × M</td>
<td>Bandwidth needed between modules</td>
</tr>
<tr>
<td>m_{CPU}</td>
<td>MI</td>
<td>M × M</td>
<td>CPU size of dependencies between modules</td>
</tr>
<tr>
<td>m_{NW}</td>
<td>Byte</td>
<td>M × M</td>
<td>Network size of dependencies between modules</td>
</tr>
<tr>
<td>A^L</td>
<td></td>
<td>Q × M × M</td>
<td>Loop module list</td>
</tr>
<tr>
<td>A^D</td>
<td></td>
<td>s × Q</td>
<td>Loop deadline</td>
</tr>
<tr>
<td>A^A</td>
<td></td>
<td>Q × 1</td>
<td>Loop application index</td>
</tr>
<tr>
<td>A^P</td>
<td></td>
<td>A × 1</td>
<td>Price for not accomplishing the application loops deadline</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>N × M</td>
<td>Nodes where each module can be deployed</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>N × M</td>
<td>Current module placement</td>
</tr>
<tr>
<td>α_{P}</td>
<td></td>
<td>1 × 1</td>
<td>Percentage of processing resources used for useful work</td>
</tr>
<tr>
<td>α_{Mem}</td>
<td></td>
<td>1 × 1</td>
<td>Percentage of memory resources used for useful work</td>
</tr>
<tr>
<td>α_{Strg}</td>
<td></td>
<td>1 × 1</td>
<td>Percentage of storage resources used for useful work</td>
</tr>
<tr>
<td>α_{Bw}</td>
<td></td>
<td>1 × 1</td>
<td>Percentage of bandwidth resources used for useful work</td>
</tr>
<tr>
<td>t_{Boot}</td>
<td></td>
<td>s × 1</td>
<td>Constant VM setup time</td>
</tr>
</tbody>
</table>
Table 3.2: Problem variables.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Unit</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$N \times M$</td>
<td></td>
<td>Binary matrix representing the module placement</td>
</tr>
<tr>
<td>$R$</td>
<td>$Z \times E$</td>
<td></td>
<td>Binary matrix representing the tuple routing map</td>
</tr>
<tr>
<td>$V$</td>
<td>$M \times E$</td>
<td></td>
<td>Binary matrix representing the module migration routing map</td>
</tr>
</tbody>
</table>

With the knowledge of the application edges, it is possible to compute several parameters related to the application modules. For instance, it is possible to compute the average required processing capacity of each module (Equation 3.3) the bandwidth between modules (Equation 3.4) and, in the worst case scenario, the quantity of instructions in the queue waiting to be processed from a given module (Equation 3.5), and the size of data that can be in the queue to be transmitted between modules (Equation 3.6).

$$m_i^{Mips} = \sum_{k \in K} \frac{e_k^{Prob} Cpu}{e_k} \cdot e_k^D = i. \quad (3.3)$$

$$m_{i,j}^{Bw} = \sum_{k \in K} \frac{e_k^{Prob} Nw}{e_k} \cdot e_k^S = i, \ e_k^D = j. \quad (3.4)$$

$$m_{i,j}^{CPU} = \sum_{k \in K} e_k^{Cpu}, \ e_k^S = i, \ e_k^D = j. \quad (3.5)$$

$$m_{i,j}^{NW} = \sum_{k \in K} e_k^{Nw}, \ e_k^S = i, \ e_k^D = j. \quad (3.6)$$

Application edges with the same source and destination can be aggregated. This results into $Z$ module pairs with dependencies, where each module pair with dependencies, $z$, is composed by its source and destination modules, $l^S_z$ and $l^D_z$, respectively.

Applications are also characterized by an arbitrary number of loops. The total number of application

Table 3.3: Characteristics of the application edges from Figure 3.3.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Periodicity [ms]</th>
<th>Probability</th>
<th>Src. module</th>
<th>Dest. module</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>10</td>
<td>1</td>
<td>sensor</td>
<td>$u$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>10</td>
<td>0.9</td>
<td>$u$</td>
<td>$v$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>10</td>
<td>0.18</td>
<td>$v$</td>
<td>actuator</td>
</tr>
<tr>
<td>$e_4$</td>
<td>100</td>
<td>1</td>
<td>$w$</td>
<td>$v$</td>
</tr>
<tr>
<td>$e_5$</td>
<td>100</td>
<td>0.4</td>
<td>$v$</td>
<td>$u$</td>
</tr>
</tbody>
</table>
loops in the system is $Q$. Each loop, $q$, belonging to application $A_q^A$, as already stated in Section 3.1, is used to specify the process-control loop of interest to the user, which may (or not) have a maximum required execution deadline, $A_q^D$. These are characterized by their list of modules composing its path within the DAG (refer to Figure 3.1), $A_q^L$. Additionally, each application $a$, can be defined by its monetary cost for not accomplishing its required loops deadline, $A_a^P$.

From the system provider or controller perspective, $D_{n,m}$ defines whether application module $m$ can be placed in node $n$. This can be used for several reasons. For instance, as stated in Section 3.2, clients can only process their own application modules. Moreover, modules can assume one of the following types: client, global or none of the previous. Nonetheless, this can also be used for security reasons. For instance, similarly to the public, private and hybrid cloud, one could implement similar fog models. Moreover, the controller, as declared in Section 3.1.1, needs to know at any time the current position of every module. Therefore, it uses $C_{n,m}$ to represent whether application module $m$, is currently instantiated in node $n$. Nonetheless, as discussed in Section 3.2, the controller assumes that nodes only use part of their resources to perform useful work. The controller defines this percentage of resource capacity for processing, memory, storage and bandwidth as $\alpha^p$, $\alpha^m$, $\alpha^s$, and $\alpha^b$, respectively. Finally, it also defines the average setup time upon migrations as $t^\text{Boot}$.

### 3.3.2 Objectives

As before mentioned, the system may not always be able to ensure every single deadline due to the hardware limitations of fog nodes. Hence, from the system provider perspective there are two main independent and conflicting objectives. On the one hand, the objective is to grant QoS to as many applications as possible. This allows users to consider it as a reliable service and, if a business model is considered as part of the study, to minimize its monetary cost for not accomplishing the defined deadlines, or even minimize the quantity of users leaving the system for not having its demands fulfilled. On the other hand, the objective is to minimize its overall energy consumption, especially when it represents the main source of monetary costs. Meanwhile, bearing in mind the QoS required to support its clients, as discussed in Section 2.3.6, it is also important to keep processor utilization as low as possible, as well as the bandwidth on links. This section presents a description of each of the formulated cost functions considered relevant and sufficient to evaluate and optimize the system performance from the system provider perspective.

- **Quality of Service Cost.** As above mentioned, one of the main objectives of the system provider is to provide QoS assurances to the highest number of client applications as possible. To do so, the system provider, based on the state of its nodes and network links, estimates the worst case latency for each application loop. Then, through the comparison between the each loop delay and the corresponding defined deadline, it verifies which application loops do not have their deadlines fulfilled. Bearing in mind that each application may have an arbitrary number of application loops, the demands of an application are considered to be fulfilled when the QoS demands of all its loops are ensured. Hence, the objective is to minimize the number of applications which have

45
one or more loop deadlines not fulfilled. This results in the quality of service cost, $C_Q$, calculated according to Equation 3.7.

$$C_Q(x) = \sum_{a \in A} c_a x_a, \quad e_a = \min \left( \sum_{q \in Q} e_q, 1 \right), \quad A_q^A = a,$$

$$e_q = \begin{cases} 1, & \text{if } L^P_q + L^T_q > A^D_q; \\ 0, & \text{otherwise}. \end{cases}, \quad \forall q \in [0, Q].$$

For each application loop, $q$, the total delay is characterized by its processing delay, $L^P_q$, and transmission delay, $L^T_q$. In order to compute these values, the $A^L_q$ matrix is followed, which defines each dependency between modules within the loop. For each generic dependency between module $i$ and $j$, the processing latency, in the worst case scenario, corresponds to the sum of the processing requirements of all tuples processed by modules hosted in the same node as module $j$ over the node processing capacity, as presented in Equation 3.8.

$$L^P_q = \sum_{i \in M} \sum_{j \in M} A^L_{q,i,j} \sum_{n \in N} P_{n,j} \sum_{l \in M} \sum_{k \in M} m_{l,k}^{CPU} \times P_{n,k}.$$

Without loss of generality, for each generic dependency between module $i$ and $j$, the transmission latency, in the worst case scenario, corresponds to the sum of the worst transmission latencies of each link comprised in the dependency path between these two modules. For each link travelled, the worst transmission latency corresponds to the sum of the network requirements of all tuples travelling through the same link over the link transmission capacity, plus the link latency, as expressed in Equation 3.9.

$$L^T_q = \sum_{i \in M} \sum_{j \in M} A^L_{q,i,j} \sum_{e \in E} R_{e} \left( \frac{\sum_{z \in Z} R_{z,e} \times m_{l,z}^{NW} \times T_{p}^D \times E_{f}^{Bw}}{\alpha^P \times E_{f}^{Bw}} + E_{e}^L \right), \quad l_{e}^I = i, \quad l_{e}^D = j.$$  

Note that in both the processing and transmission delays, the capacity of the nodes and the links is multiplied by a factor $\alpha^P$ and $\alpha^b$, respectively, (defined by the system provider) which defines the percentage of available resources effectively allocated to perform useful work (i.e., not used for operational maintenance operations).

- **Power Cost.** From the system provider perspective, as above mentioned, is also important to minimize the energy consumption once it represents the main source of monetary costs. The power consumption cost, $C_{Pw}$, is related both to the processing cost, $C_p$, and to the transmission cost of tuples over the cellular network, $C_B$. The processing power consumption, follows the linear power consumption formulation presented in Equation 3.1. Therefore, it represents the
multiplication between the percentage of processor utilization and the power consumption when using the full processor capacity. Similarly, the transmission power cost represents, for each link, the multiplication between the power consumption when using the full transmission capacity and the percentage of link utilization. The power consumption cost is presented in Equation 3.10.

\[
C_{Pw}(x) = C_P(x \times (f^{Pw} - f^{Pw})) + C_B(x \times f^{Txe}).
\] (3.10)

**Processing Cost**

Bearing in mind the QoS required to support its clients, it is also important to keep processors utilization as low as possible. This raises the processing cost, presented in Equation 3.11.

\[
C_P(x) = \sum_{n \in N} f_{_Fog}^n x_n \frac{P_n \times m^{Mips}}{\alpha^P \times f^{Mips}_n}.
\] (3.11)

In this cost, \(P_n \times m^{Mips}\) represents the total amount of processing capacity required for all modules placed in node \(n\). Similarly to Equation 3.9, the processing capacity of node \(n\) is considered to be only a part of its full capacity, \(f^{Mips}_n\), by multiplying it by \(\alpha^P\). Moreover, as already stated, clients have some differences compared to fog or cloud nodes regarding the objectives of the system provider. In this particular case (i.e., the processing cost), as the clients already have advertised the amount of resources that they are willing to use, provided that the constraints are fulfilled, the system provider is not motivated to minimize its utilization. In fact, it is willing to use the client nodes as much as possible once it represents no power consumption costs for itself. Therefore, the binary system variable \(f_{_Fog}^n\) is introduced, where \(f_{_Fog}^n = 0\) means that node \(n\) is a client. Note that this affects both processing and power costs.

• **Bandwidth Cost.** The bandwidth cost, presented in Equation 3.12, is very similar to the processing cost.

\[
C_B(x) = \sum_{i \in Z} m^{Bw}_{i_{_Fog} x_i} \sum_{e \in E} f_{i_{_Fog} x_i} R_{z,e} \frac{R_{z,e}}{\alpha^b \times E^{Bw}_e}, \quad i = E^S.
\] (3.12)

In this cost, \(\sum_{z \in Z} R_{z,e} \times m^{Bw}_{i_{_Fog} x_i}\) gives the amount of bandwidth resources required on link/edge \(e\) to support every dependency between pair of modules while using a given matrix \(R\). Therefore, similarly to the processing cost, the objective is to minimize the result of required bandwidth resources over the quantity of available ones. Note that, similarly to Equation 3.11, the binary system variable \(f_{i_{_Fog}}^n\) is introduced in order to set to zero the cost of transmitting tuples from client nodes. This is motivated by the fact that client nodes can only process their own modules and are only connected to a static fog or cloud node. Therefore, a client node never acts as a bridge of transmission between two nodes. In this way, the system provider does not have to minimize its bandwidth usage nor the energy consumption inherent to its use.
• Migration Cost. Migrations are an additional overhead to the system, which are included within the operational maintenance. Although some bandwidth is reserved to perform migrations \((1 - \alpha b)\), as discussed in Section 3.2, these still affect the applications of other users due to the fact that network links only transmit a tuple at a time. Therefore, the goal is to use the links which have higher available bandwidth in order to minimize its impact. In this way, the migration cost, presented in Equation 3.13 follows the same principle of processing and bandwidth costs. Note that the migration, as already mentioned, is performed using full VM migration. Therefore, the migration size accounts with the total VM size \((i.e., m_{Strg}^m + m_{Mem}^m)\).

\[
C_M(x) = \sum_{m \in M} (m_{Strg}^m + m_{Mem}^m) \sum_{e \in E} f_{Fog}^x \frac{V_{m,e}}{(1 - \alpha b)} \times \frac{E_{Bw} i}{E_{S} e}, i = E_{S} e. (3.13)
\]

• Operational Cost. One of the goals of this work was to provide a flexible problem formulation. For instance, instead of optimizing the objectives from the system provider perspective, one might aim to implement it from the broker perspective. This means that, instead of owning the whole physical infrastructure, the broker relies in one or more third party fog and cloud infrastructure providers in order to rent the required hardware. In this case, aside from providing QoS, the broker would also aim to maximize its profit.

From the broker perspective, the computing resources can be acquired with different prices (depending on the different owners imposed values). Assuming that it sells these computing resources at fixed prices per unit to its clients, in order to maximize its profit, it needs to find the best combination of resources, allowing to minimize its monetary cost and yet met its users QoS demands.

In order to formulate the problem from the broker perspective, every (above described) cost formulation receives a given parameter \(x\). This parameter represents the different prices for each resource utilization in each hired node. Therefore, in such scenario, the operational cost formulation would be as presented in Equation 3.14.

\[
C_O = P_{Mips} \times P \times m_{Mips} + P_{Mem} \times P \times m_{Mem} + P_{Strg} \times P \times m_{Strg} + \left( P_{Bw} \sum_{z} m_{Bw}^z \times R_z \right) 1N + C_{Pw} \left( (P_{Pw})^T \right) + C_Q(A^P) (3.14)
\]

For instance, in this formulation, the operational cost for the case of processing utilization is given by \(P_{Mips} \times P \times m_{Mips}\). The same principle is applied to compute the price for memory, storage and bandwidth resource utilization. Note that \(1N\) corresponds to a \(N \times 1\) matrix filled with the value 1. Then, \(C_{Pw} \left( (P_{Pw})^T \right)\) would be used to pass a given power consumption vector price \(P_{Pw}\). Similarly, each application whose requirements cannot be respected, \(a\), could have an associated monetary cost, \(A^P_a\) which would be used in the quality of service cost formulation as \(C_Q(A^P)\).

Note that, in this case, all prices on client nodes would be zero. Also, note that the migration cost
is not considered in this formulation because all the presented terms in Equation 3.14 are steady state average values, while the migration is a transient cost. Finally, note that this formulation is applied in scenarios where the broker uses a business PAYG model (i.e., a payment method which charges based on usage).

3.3.3 Constraints

Although the aim is to optimize the system provider objectives, there are some constraints which need to be respected. This section presents the formulation and description of each constraint.

- **Problem Variables Type.** The first constraints are related with the type of problem variables. As already mentioned, all problem variables are binary. Hence, the definition of the possible values for the placement of application modules, \( P \), the data dependency routing matrix, \( R \), and the VM migration routing table, \( V \), are presented in Equations 3.15, 3.16, and 3.17, respectively.

\[
P_{i,j} \in \{0, 1\}, \forall i \in [0, N], \forall j \in [0, M]. \tag{3.15}
\]

\[
R_{z,e} \in \{0, 1\}, \forall z \in [0, Z], \forall e \in [0, E]. \tag{3.16}
\]

\[
V_{m,e} \in \{0, 1\}, \forall m \in [0, M], \forall e \in [0, E]. \tag{3.17}
\]

- **Resource Utilization.** Regarding the resources, it is required to ensure that no resource utilization exceeds the available resources. This is applied for every resource including the processor, memory, storage, an bandwidth, resulting in Equations 3.18, 3.19, 3.20, and 3.18, respectively. Note that, in each resource, a percentage should be reserved to perform operational maintenance operations and other overheads. Additionally, in order to avoid divisions by 0, every node processing capacity and every edge communication bandwidth should by higher than 0, resulting in Equations 3.22, and 3.23, respectively.

\[
P \times m^{Mips} \leq \alpha^P \times f^{Mips}. \tag{3.18}
\]

\[
P \times m^{Mem} \leq \alpha^m \times f^{Mem}. \tag{3.19}
\]

\[
P \times m^{Strg} \leq \alpha^s \times f^{Strg}. \tag{3.20}
\]

\[
\sum_{z \in Z} m_{z,e}^{Bw} \times R_{z,e} \leq \alpha^b \times E_e^{Bw}, \forall e \in [0, E]. \tag{3.21}
\]

\[
f^{Mips}_n > 0, \forall n \in [0, N]. \tag{3.22}
\]

\[
E_e^{Bw} > 0, \forall e \in [0, E]. \tag{3.23}
\]

- **Possible Deployment.** Equation 3.24 ensures that each application module is placed in one and only one node. Equation 3.25 ensures that each application module is placed in a valid node.
Figure 3.4: Directed graph example used to compute the shortest path between two nodes.

\[
\sum_{n \in N} P_{n,m} = 1, \forall m \in [0, M].
\]  \hspace{1cm} (3.24)

\[
P \leq D.
\]  \hspace{1cm} (3.25)

• **Routing Paths.** Equations 3.26 and 3.27 express the data dependencies and migration routing paths, respectively.

\[
\sum_{i \in E} R_{z,i} - \sum_{j \in E} R_{z,j} = P_{n,i}^z - P_{n,j}^z, \forall z \in [0, Z], \forall n \in [0, N],
\]  \hspace{1cm} (3.26)

\[
E_{i}^S = n, E_{j}^D = n.
\]

\[
\sum_{i \in E} V_{m,i} - \sum_{j \in E} V_{m,j} = C_{n,m} - P_{n,m}, \forall m \in [0, M], \forall n \in [0, N],
\]  \hspace{1cm} (3.27)

\[
E_{i}^S = n, E_{j}^D = n.
\]

These routing paths are computed similarly to the how weighed shortest path problems are computed. However, in this case, weights appear due to the influence of the defined objectives. For instance, let us consider the directed graph presented in Figure 3.4.

In order to compute the shortest path between any two nodes, using a linear programming problem formulation, it is possible to represent the graph in a weight vector, \(w\), a source vector, \(s\), and a destination vector, \(d\). For instance, considering \(e_1\) to be the edge from node \(A\) to node \(B\), then \(w_1 = W1, s_1 = A\), and \(d_1 = B\). Note that each edge is directional, therefore, between node \(A\) and \(B\) there exists the edge \(A \rightarrow B\) and \(B \rightarrow A\). In this way, for a generic directed graph with \(N\) nodes and \(E\) edges, the shortest path can be computed using the problem formulation presented in Equation 3.28, where \(S\) represents the source node and \(T\) represents the target node. In Equation 3.28c, it is defined that for a given node \(n\), the sum of all its outgoing edges minus the sum of all its incoming edges must be 1 when \(n\) is the source node, \(S\), −1 when \(n\) is the target node, \(T\), or 0.
when it is not neither of them.

\[
\begin{align*}
\min_x & \quad \sum_{e \in E} w_e x_e \quad \text{(3.28a)} \\
\text{subject to} & \quad x_e \in \{0, 1\}, \forall e \in [0, E] \quad \text{(3.28b)} \\
\sum_{i \in E} x_i - \sum_{j \in E} x_j &= \begin{cases} 
1, & \text{if } n = S; \\
-1, & \text{if } n = T; \\
0, & \text{otherwise.} 
\end{cases} \quad \text{(3.28c)}
\end{align*}
\]

Regarding Equation 3.27, \( C_{n,m} - P_{n,m} \) expresses the same operation as the one presented in Equation 3.28c. For instance, if some module \( m \) needs to be moved from node \( n' \) towards \( n'' \), \( C_{m,n'} - P_{m,n'} = 1 \), \( C_{m,n''} - P_{m,n''} = -1 \) and \( C_{m,n} - P_{m,n} = 0 \), \( \forall n \neq n', \forall n \neq n'' \). The same principle is applied to the Equation 3.26.

- **Migration Deadline.** Finally, the last constraint regards to the application modules migration deadline. As aforementioned, each application module can be defined with a given migration deadline, which should not be exceeded, as expressed in equation 3.29. In contrast to all the above described constraints, this is only applied when recomputing the decision-making algorithm. This means that, in the first execution of the algorithm (i.e., before the simulation has started) it is not applied.

\[
L^M_m \leq m^{M_{\text{migD}}}, \forall m \in [0, M]. \quad (3.29)
\]

As already discussed, migrations are included in the operational maintenance operations. Therefore, in order to compute the migration time, it is considered only the bandwidth reserved for this kind of operations, as it is presented in equation 3.30. Additionally, VMs have a given setup time, \( \ell^{\text{boot}} \), which is also considered in this formulation. In this case, the binary variable \( b \) is 1 if and only if the node currently hosting the VM, \( m \), is not the same as the one given by the placement matrix \( P \).

\[
L^M_m = b \times \ell^{\text{boot}} + \sum_{e \in E} \ell_{m,e} \left( \frac{m^{S_{\text{mem}}} + m^{M_{\text{mem}}}}{(1 - \alpha^b) \times E^{\text{bw}}_e + E^L_e} \right), \quad b = C_{n,m} - P_{n,m}, \quad C_{n,m} = 1. \quad (3.30)
\]

### 3.3.4 Concluding Remarks

From a system provider perspective, there are some independent and conflicting objectives. For instance, the power cost aims to minimize the energy consumption while the processing cost aims to minimize the percentage of processors utilization. For instance, in IoT environments, presumably there are nodes with low power consumption offering low processing resources. Therefore, power cost aims
to deploy as many modules as possible in these nodes, while processing cost aims to deploy evenly (i.e., based on its processing capacities) the modules between the nodes. Moreover, minimizing processing and bandwidth cost are also conflicting objectives. On the one hand, processing cost aims to achieve a weighed workload balance, while the bandwidth cost aims to use as little as possible the network links, as well as to achieve a weighed network load balance. Thus, there exists a trade-off between these two costs. As more dependent modules (i.e., modules having some application edge connecting them) are deployed inside the same node, less bandwidth has to be used, however more overloaded the node becomes, and vice-versa. Also, as above discussed, the QoS and the power consumption are also conflicting objectives. This way, from the system provider perspective, the problem is defined as a multi objective problem presented in Equation 3.31. Meanwhile Equation 3.32 presents the problem formulation from the broker perspective. In either case, the final problem formulation is given in Equation 3.33.

\[
F = \left[ C_Q(1N), C_{Pw}(1N), C_P(1N), C_B(1N), C_M(1N) \right]^T
\]

\[
F = C_O
\]

\[
\min_{F,R,V} \quad F
\]  

\[
\text{subject to} \quad (3.15) - (3.17) \quad \triangleright \text{Problem variables type} (3.33b)
\]

\[
(3.18) - (3.23) \quad \triangleright \text{Resource utilization} (3.33c)
\]

\[
(3.24) - (3.25) \quad \triangleright \text{Possible deployment} (3.33d)
\]

\[
(3.26) - (3.27) \quad \triangleright \text{Routing paths} (3.33e)
\]

\[
(3.29) \quad \triangleright \text{Migration deadline} (3.33f)
\]

In order to solve multiple objective problems, there are two likely solutions: (1) to use a weighted sum problem formulation, and (2) to use priority-based problem formulation. The first one (i.e., weighted sum) was proposed in several reviewed works (e.g., [41], [42], [44]). However, as the defined costs have different units and order of magnitude, it would increase the problem complexity and ambiguity. It would require to use some kind of normalization technique and to use weights to vary the impact of each cost. These are more subjective and, in the presence of different units, more prone to misleading conclusions. Therefore, the proposed problem formulation as well as the algorithms used to solve it, presented in Section 3.4, are priority-based.

One of the objectives of this work was to compare the solutions found by several optimization algorithms. As it will be shown in Section 3.4, one of them is performed by the CPLEX optimizer [87], which is capable of finding the optimal problem solution. Unfortunately, in its current version, V12.9.0, it does not allows to implement quadratic formulations both for objectives and constraints while defining multiple objective problems [88]. Also, it does not allows to implement divisions by problem variables. Therefore,
in the processing and bandwidth cost formulations it was not possible to directly express fairness (e.g., Jain’s fairness index) among the entities. If such had been implemented, it could allow to achieve the resource balance between the entities more evenly.

Regarding Table 2.3, the proposed problem formulation covers the QoS, bandwidth, and power costs. Additionally, it also covers the processing and migration costs. As aforementioned, operational cost is not one of the objectives of the system provider. Nonetheless, bearing in mind the broker perspective, the proposed problem formulation is flexible enough to allow this cost to be easily implemented. It should be noted that the QoS could also be easily changed into a constraint. In such case, the system provider could either advertise the client that its deadline demands are not ensured, or otherwise remove it from the system. Moreover, all placement, data dependency and migration routing variables are considered. Regarding the constraint, resources, bandwidth, and migration deadline are also defined. As already mentioned, in Section 3.2, power is not a constraint once it is assumed that mobile nodes advertise the amount of resources which they are willing to use. Additionally, the possible deployment matrix, $D$, is implemented which, as above discussed, can have multiple advantages. Finally, the optimization is performed in a centralized manner. Distributed architectures leading to higher scalability are relegated to future work.

### 3.4 Simulation Architecture

In order to implement the proposed architecture, this work uses the iFogSim simulator as its basis. It, as presented in Table 2.4, has already several built-in features which meet the objectives of this work. However, it presents some drawbacks such as not providing communication between fog nodes at the same hierarchical level, and assuming that both fog nodes and end devices are static. In order to cover all features presented in Table 2.2, some major changes ought to be performed in this simulator at the scenario design, system optimization and result levels. This section presents a review of these changes and why they were performed.

#### 3.4.1 Scenario Design

The scenario or architectural design level, as discussed in Section 3.1, is structured by entities and services. The iFogSim simulator already has several built-in features in this regard. Looking through Table 2.2, this simulator, with respect to the physical infrastructure, already allows to implement several cloud, fog, and client nodes, and to benefit from their cooperation (i.e., nodes can communicate and work together in order to enhance fog services QoS). Meanwhile, regarding the services themselves, it already supports the implementation of applications structured as a collection of modules using DDF programming model. Moreover, it allows the use of VMs in order to encapsulate application modules, allowing nodes to host several modules at the same time. Nonetheless, it still has some shortcomings. For instance, the implemented cooperation is limited due to the fact that it does not allow communication between fog nodes at the same hierarchical level neither implements mobility nor location awareness.
mechanisms of fog and client nodes. Therefore, it does not allow implementing opportunistic mecha-
nisms such as migration of services in order to further enhance users’ services QoS in the presence of
mobile environments.

Entities

As above mentioned some major changes had to be performed with respect to the entities and how
communication is performed. This section describes each modification and why it had to be performed.

• **Transform the hierarchical architecture into a directed graph architecture.** As aforemen-
tioned, the iFogSim simulator allows implementing hierarchical architectures. In practice, each
node can be connected to one upper level node (i.e., its parent node) and to many lower level
nodes (i.e., child nodes). Therefore, it does not allow the communication between two nodes at the
same hierarchical level. For instance, in that figure, two nodes at the edge of the network (e.g., $F_2$
and $F_3$) could be connected to the same parent node ($F_1$) and yet been connected to each other,
allowing to further enhance the QoS offered to its hosted application modules deployed by clients
($C_1$ - $C_4$). In order to generalize the architecture and implement arbitrary connections between
any two nodes, this feature had to be changed. To do so, a directed graph architecture approach
was adopted. In this way, it is possible to implement any potential scenario without restrictions in
the number of nodes, number of connections and the nodes involving these connections. Also,
this allows to deploy modules in any node instead of being restricted to the path between the
client (lower hierarchical level) and the cloud (higher hierarchical level). Nonetheless, this allows
to remove some node characteristics which in practice are meaningless such as its level, and the
up (towards the cloud) and down (towards the client) directions defining the relationship between
parent and child nodes;

• **Change the communication model.** The iFogSim simulator performed its communications from
a parent node to its child nodes and from a child node to its parent node using different methods.
For instance, each node was defined with a given bandwidth available to communicate with its
parent node. Similar to the proposed architecture, it also uses a tuple network queue in order to
transmit one tuple at a time. Meanwhile, each node was also defined with a given bandwidth used
to communicate with its child nodes. Then, whenever a given tuple was required to be transmitted
to a child node, the tuple was sent in broadcast to all its child nodes. Note that, although sent in
broadcast, this operation was performed by transmitting the tuple to one child node at a time. In
this case, the bandwidth used to compute the tuple transmission time interval was the same to all
child nodes. It is worth noting that, each node could only transmit one tuple to its parent node and
to one of its child nodes at the same time.

There are some flaws in this communication model. First, the communication between nodes in
fog environments can assume a wide range of technologies. Therefore, different communication
links can be characterized by different transmission velocities. Second, the transmission of tuples
should not be performed in broadcast. As different nodes host different modules, and tuples are
characterized by their destination modules, tuples should be sent in a directional manner in order to remove such network overhead. Finally, instead of using the same tuple network queue to all its child nodes, each connection should have its own tuple queue. Again, in order to generalize the architecture, this feature had to be changed. To this end, while using the directed graph architecture approach, the bandwidth available is defined at the network links instead of being defined at nodes themselves. In this approach, network links are also characterized by its source and destination nodes (it is assumed a full-duplex communication) and its communication latency. As already discussed in Section 3.2, both bandwidth and latency are used to mask the network details. Moreover, routing tables were also added to the fog devices in order to submit the required tuples to the right network link tuple queue which, in turn, is responsible to forward the tuple to the right network link;

- Add mechanisms to support mobility of nodes. The iFogSim simulator assumes that all nodes are static. However, assuming completely static fog environments is unrealistic. Therefore, the proposed architecture also addresses mobility and location awareness. To this end, each node is defined with a given type of movement. In this work three types of movement were implemented, namely static, random, and periodic. The first one (i.e., static) is defined only by its location in a two-dimensional plane. This movement type can be used by any node. For instance, cloud nodes are clearly static nodes. Moreover, some fog and client nodes can also assume this type of movement. The random type of movement is defined by a given current position, its random velocity and direction. In the proposed architecture, the random movement can assume four different velocities and eight different directions (N, NE, E, etc.). Then, following a given update time interval, the current position is calculated based on the previous positions in the plane, the velocity and direction, with the latter being updated using an arbitrary probability distribution. Meanwhile, the periodic movement is characterized by a rectangular path (can assume the form of a rectangle, square or a line) which is repeated throughout the simulation with a constant predefined velocity. The latter was motivated by the work presented in in [39], which highlights the use of buses as mobile fog nodes due to their fixed mobility trajectories and strong periodicity;

- Add the cellular network. In order to support the mobility of nodes, some form of mobile communication model had to be implemented. As already discussed in Section 3.2, this is performed using cellular networks which are available in the whole 2D plane. Moreover, as also discussed, each mobile node needs to be always connected to its closest static fog or cloud node in order to achieve a fully interconnected network of nodes. Therefore, the handover mechanism was also implemented, which includes both the creation and the removal of connections between nodes;

- Change the processing model. As already discussed in Section 3.2, the iFogSim simulator allows processing several tuples inside the same node at the same time while dividing equally the CPU capacity among them. However, the number of processed MIs throughout the simulation does not match the number of required MIs to be processed. This unexpected behavior would result in the decision-making algorithm giving apparent wrong solutions. Therefore, similar to the
network links, a queue is implemented in order to process only one tuple at a time;

- **Change the pricing model.** Bearing in mind the broker perspective, the proposed architecture also implements a different pricing model from the one presented in the iFogSim simulator. Aside from considering the utilization of processing resources, as defined in the iFogSim, regarding the problem formulation presented in Section 3.3, the proposed pricing model also takes into account the utilization of the other resources (i.e., memory, storage and bandwidth), as well as the energy consumption;

- **Change the energy model.** Similarly to the pricing model, the iFogSim simulator only implements the linear power consumption model presented in Equation 3.1 as a source of energy consumption in the nodes. Meanwhile, in the proposed architecture, as discussed in Section 3.3, the power consumption for mobile communications is also considered in this computation.

**Services**

Similarly to entities, the services or applications have also suffered some major changes in order to meet the objectives of this work. These are listed and described below.

- **Add different types of application modules.** As already mentioned in Section 3.2, the proposed architecture introduces three different types of application modules, namely: client, global or normal (i.e., none of them). In this way, it is possible to implement multi-player or single-player applications. For instance, if all modules in one user application are of type client or normal, the application is single-player. Therefore, all modules of that application are placed in order to optimize the QoS perceived by the client. Meanwhile, if the application is constituted of one or more global modules, the application is said to be multi-player. In this case, the placement of these global modules is optimized for all clients using them. This approach is motivated due to the fact fog computing is supposed to be able to support any type of application. However, the iFogSim simulator only allows for multi-player applications;

- **Add the possibility to deploy multiple applications.** Each client is provided with the possibility to deploy multiple applications (single or multi-player) instead of being restricted to only one multi-player application as it is in the iFogSim simulator. For that purpose, users can use (or not) one pair of sensor and actuator for each application;

- **Add application loops deadline.** Although the iFogSim already defines application loops, in order to specify the process-control loops of interest to the user, it does not allow to define a given deadline. As the fog computing is intended to support real-time applications, this is an important feature. In the proposed architecture the possibility of defining (or not) the application loop deadline is provided in order to optimize the system performance while respecting the users imposed QoS;

- **Add the computation of the required resources to support the application modules.** As already discussed, in Section 3.4.2, both the processing and bandwidth resources required to
support each application module are computed based on characteristics of the latter (due to the assumption that these are known);

• **Add the VM migration mechanism.** As the proposed architecture considers mobile environments, the system needs to be adjusted whenever needed. To do so, the possibility to migrate VMs between nodes is added. As already stated in Section 3.2, migrations are performed through full VM migration which can be defined (or not) with a given migration deadline. This mechanism was not implemented in the iFogSim simulator, since it only considers static environments. Meanwhile, as already discussed in Section 2.4, MyiFogSim had implemented this mechanism using a rough approach. In the proposed architecture, this mechanism considers both the network and server states as well as the impact it causes in its own and other users applications.

**Graphical User Interface**

In order to ease the continuity of test and development by other research teams using the proposed architecture, a GUI was developed. In this way, instead of defining the entire architecture programmatically, researchers can define it in a user-friendly environment. The GUI implemented in the iFogSim simulator only allows to define the physical infrastructure (with few parameters), not allowing to define the applications neither to run the simulation from it. Therefore, some major modifications were performed. Figure B.1 depicts some of the implemented features. These include the add, edit and removal of physical entities, network connections, applications and all its components, define the applications which each user aims to deploy into the fog system, generate random scenarios, adjust the simulator settings, read and write the scenarios using JavaScript Object Notation (JSON) files, and run the simulation.

**3.4.2 Optimization**

In order to optimize the system, the controller is responsible of finding the module placement and both dependency and routing tables based on the current system state and the multiple objective problem formulation defined in Section 3.3. To do so, four different algorithms using priority-based optimization were implemented, namely: CPLEX optimizer, Brute Force Algorithm (BFA), GA and RA. The current section presents how these algorithms were implemented and how solutions are evaluated.

**CPLEX Optimizer**

In order to find the optimal solution, the problem formulation can be solved using the CPLEX optimizer - a high-performance mathematical programming solver for linear programming, mixed-integer programming and quadratic programming, which conventionally terminates as soon as it finds an optimal solution [89]. However, it does not allow implementing quadratic programming while defining multiple objective problems. Hence, some workarounds were adopted in order to be able to implement the proposed problem formulation presented in Section 3.3. For instance, regarding the QoS cost formulation, presented in Equation 3.7, a multiplication between two problem variables both in Equation 3.8 and Equation 3.9.
is implemented, resulting in a quadratic formulation. However, as both multiplications are performed between binary variables, these can be replaced with the function \( \min \). This function is provided by the CPLEX optimizer itself, which returns an expression representing the minimum of its two arguments. For instance, in Equation 3.8, the numerator (i.e., \( P_{n,j} \times \sum_{l \in M} \sum_{k \in M} m_{l,k}^{CPU} \times P_{n,k} \)) intends to sum \( m_{i,k}^{CPU} \) when both \( P_{n,j} = 1 \) and \( P_{n,k} = 1 \). In this way, it can be replaced with the following expression: \( \sum_{l \in M} \sum_{k \in M} m_{l,k}^{CPU} \times \min (P_{n,j}, P_{n,k}) \). The same principle is applied to the Equation 3.9. The remainder of the formulation is defined (using the available CPLEX methods) as it is presented in Section 3.3.

While defining multiple objective problems using the CPLEX optimizer, it is possible to define four parameters. For each objective, it is possible to define its priority, weight and relative and absolute tolerances. Then the CPLEX multiple objective optimization algorithm sorts the objectives by decreasing priority value. If several objectives have the same priority, they are blended in a single objective using the provided weight attributes. As a result, CPLEX constructs a sorted list of objectives (or blended objectives), each with a unique priority. CPLEX can then proceed to find the lexicographically minimal (or maximal) solution for this order. To obtain this solution, each objective is optimized in turn by decreasing order of the priority value in a hierarchical manner. Whenever the optimal solution for an objective (or blended objective) is found, CPLEX imposes that, for the remaining (lower priority) objectives, the only solutions considered are those that are also optimal for the previous (higher priority) optimized objectives. Finally, for each objective, the relative and absolute tolerances specify the maximum deviations allowed from the optimal value of that objective [90]. As the defined costs are characterized by different units and different orders of magnitude, each objective is always defined with a different priority in order to avoid the weight sum. In this way, depending on the type and size of the implemented architecture, the solution might possibly be restricted to its highest priority (i.e., the highest priority is optimized and the remaining objectives have their search space quite restricted). Aside from that, it is also difficult to define the absolute tolerances for each objective once it depends on the implemented scenario and the current system state. Therefore, in the interest of reducing the priorities impositions, relative tolerances are used, where each objective is defined with an arbitrary value.

**Brute Force Algorithm**

With the intention to validate the remaining algorithms, its solutions and execution time, the BFA was also implemented. Similarly to the CPLEX optimizer, it is also able to find the optimal solution, in this case by searching in the whole search space. This algorithm is implemented using the typical recursive brute force approach, which is presented in Algorithm 1. This algorithm, recursively starts to fill the placement matrix, \( P \). While doing so, it verifies whether the selected node \( n \) can host the selected module \( m \) by verifying if it respects the possible placement matrix, \( D \). If it does respect \( D \), it keeps filling \( P \), otherwise it selects the next node. When \( P \) has finally been filled up (with one possible solution), it verifies if no node resources are exceeded. If the resources are respected, it starts solving the routing. As already mentioned, there are two routing matrices which need to be found: the data dependency routing, \( R \), and the migration routing, \( V \). In all algorithms except the CPLEX optimizer, these are not binary matrices. In order to ease the implementation, although rows have the same meaning (e.g., each
row of $R$ represents the path followed by the corresponding dependency), each column represents a node instead of an edge. With this approach, in the worst case scenario, if the physical topology has $N$ nodes, paths can be defined with $N$ different nodes. Nonetheless, as soon as the final node is reached in a given path, the remaining columns of that row are automatically filled with that node.

Still, in Algorithm 1, in order to start solving the routing paths, an empty $R$ matrix is created, and each initial and final position is filled based on matrix $P$. Therefore, the objective is to find the best paths given each initial and final nodes. To do so, it is applied the same principle as the one used to fill the matrix $P$. The difference lies in the verification of whether the selected node $n$ is a valid node or not. To be considered valid, $n$ cannot be already in the path, needs to be connected to the previous node and still represent a distance to the final node no longer than the remaining number of hops which is a value precomputed by the Dijkstra’s algorithm while setting all weights to 1. Then, as already mentioned, as soon as final node is reached in a given path, the remaining hops are repeated with that node. When $R$ has finally been filled up (with one possible solution), the process is repeated in order to fill the matrix $V$.

Finally, in order to compare the solutions, the pseudocode presented in Algorithm 2 is used. This procedure is similar to the one performed in the CPLEX optimizer. In this case, the number of violated constraints perform the rule of the highest priority cost with no relative tolerance. Then, for each objective, in decreasing order of priority, the costs of both solutions are compared using a given relative tolerance.

**Random Algorithm**

In contrast to the previous algorithms, the Random Algorithm (RA) was implemented in order to achieve near-optimal solutions. Its execution is depicted in Algorithm 3. This procedure consists in generating pseudo-random solutions, without knowledge of the previously generated solutions, while using two stop conditions. On the one hand, the maximum allowed number of iterations which the algorithm can execute is defined. On the other hand, the number of iterations in which it is considered that the algorithm has converged is defined. The latter is performed through the use of a counter which is incremented when, between iterations, the $bestSolution$ does not change significantly. On the contrary, whenever is does, this counter is set to 0. As it can be seen in Algorithm 3, for each iteration it generates a given pseudo-random matrix $P$. This procedure is presented in Algorithm 4. As can be noted, it always respects the defined matrix $D$. Then, based on the generated matrix $P$, the two routing matrices $R$ and $V$ are created. Similarly, to the described in the BFA, in these matrices, all source and destination nodes are filled up. Afterwards, both matrices are filled up pseudo-randomly using the pseudocode defined in Algorithm 5. Again, similarly to the described in the BFA, this procedure goes for each column in a given path, and verifies if the destination was already been found. If so, the column is filled with the destination node. Otherwise, similar to the generation of the pseudo-random matrix $P$, it creates a list of possible nodes. These correspond to the nodes being connected to the previous one and representing a distance to the final node no longer than the remainder number of hops. Then a random node is selected from the set of possible nodes.
Last but not least, the Genetic Algorithm (GA) was also implemented in order to achieve near-optimal solutions. In this case, in contrast to the RA, it generates heuristic solutions using the knowledge of the previously generated solutions. Moreover, similarly to the BFA and RA, it was also implemented using a two-step algorithm (i.e., both $R$ and $V$ matrices are computed based on matrix $P$). This is motivated due to the fact that changing some placement in the matrix $P$, invalidates several elements in both matrices $R$ and $V$. The two-step GA is depicted in Algorithms 6 and 7. Similarly to the RA, in each of the two-step algorithm, two stop conditions are used, namely: the maximum number of generations and the number of generations in which it is considered that the algorithm has converged.

As illustrated in Algorithm 6, at the beginning a new population with an arbitrary number of individuals, is generated, each of which with a pseudo-randomly generated matrix $P$. Then, for each iteration, each individual receives the best matrices $R$ and $V$ based on its matrix $P$ through the Algorithm 5. Afterwards, if the convergence stop condition is not yet accomplished, the new generation is created. For that purpose, the three genetic operators are used. The first is the selection operator, which gives preference to the individuals with good fitness scores and allow them to pass their genes to the successive generations. In this way, the matrices $P$ of a given percentage of the best individuals ($FITTEST$), is passed to the new individuals in the new generation. Then the crossover operator represents mating between individuals. Two individuals are selected using selection operator and crossover positions are chosen randomly. Then, the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). To do so, for each of the remaining new individuals, two random individuals are selected from the set of the best individuals (with size $POPULATION_SIZE * p$) and mating between them is performed. This procedure is presented in Algorithm 8. As it can be seen, for each module, $m$, using a random operation, the new individual may be defined with the genes from either of its parents. Additionally, the mutation operator is also implemented, which inserts random genes in offspring to maintain the diversity in population to avoid the premature convergence. In this case, if none of the parent genes passes to the child chromosome while defining the genes for module $m$, the mutation is used to generate a random placement which respects the matrix $D$.

Meanwhile, regarding the procedure used to generate both the $R$ and $V$ matrices, as depicted in Algorithm 5, for each individual present in the received population, a GA is executed. In this case, for each GA, the new population is always characterized with the same matrix $P$ as the corresponding received individual. Then, the $R$ and $V$ matrices are filled with its corresponding start and destination nodes based on matrix $P$. The remaining of this procedure is very similar to the one presented in the Algorithm 6. The main differences lie in the mating between the selected individuals. In this case, the implemented procedure is depicted in the Algorithm 9. Similarly to the mating performed in the Algorithm 8, this also has a given probability, for each path, of being defined by the genes from either of its parents. Additionally, the mutation operator is also implemented, which defines a random path based on its start and destination node.
3.4.3 Results

Finally, in order to evaluate the system performance, some modifications to the iFogSim simulator were performed, which concern the statistics of both the algorithm and the simulation execution itself. The implemented modifications allow the analysis several statistics. Regarding the optimization algorithm, it is possible to verify the computed cost of each objective of the best solution, the worst case scenario delay for each application loop, the plot between the objectives cost value and the corresponding iteration, and its execution time. Meanwhile, for the simulation, analyse its execution time, the QoS of each application loop (including the minimum, maximum, average and standard deviation delay, as well as the number of completed and violated loops), the QoS of each application module during migrations (including the minimum, maximum, average and standard deviation delay, as well as the number of completed and violated migrations). Also, it allows to verify the energy consumption and the CPU utilization in each node and the total for the system provider nodes, as well as the network usage in each link and the total for the system provider links. Nonetheless, regarding the mobile scenarios, it is possible to verify the number of tuples delivered successfully and dropped/lost, and the number of handovers.

3.5 Summary

In this chapter, the proposed architecture, the problem formulation, the algorithms to solve the problem formulation and the simulation model have been presented in detail. Fog computing is a computing paradigm assuming a quite generic nature. It is structured of a network of interconnected nodes with processing capabilities, using a wide range of communication technologies. The nodes are deployed between the end users and the cloud, and they intend to offer the possibility to support real-time applications. The generic nature of fog computing lies in fact that it does not rely in any specific designed and purposely built technology or aims to support a specify type of application. Therefore, the proposed architecture, in contrast to the previously proposed and reviewed ones, assumes a generalized form regarding all nodes, communications and applications. Additionally, bearing in mind dynamic environments, the proposed architecture also assumes the presence of mobile fog and client nodes. Although introducing new challenges such as migration of services between nodes, it can be explored in order to further enhance the QoS perceived by the users. Afterwards, provided that constraints (e.g., time boundaries) are fulfilled, its up to the orchestrator, based on its predefined objectives, to decide which node should host and execute each user service. Regarding the proposed solution, this decision-making can be executed using several optimization algorithms such as the CPLEX optimizer, brute force, random or genetic algorithm. In order to simulate different scenarios and different decision-making algorithms, the proposed solution has further extended the iFogSim simulator capabilities. Finally, results regarding the algorithm and the simulation itself are collected throughout the simulation and manipulated through appropriate operations to extract relevant QoS metrics to both the users and the system provider.
Chapter 4

Results

In this chapter, the performance of the proposed algorithms is assessed in a set of different scenarios configured in the developed fog computing simulator. For this purpose, first, a set of simulations using with different static and location unaware scenarios are analysed. These allow to compare the performance of the algorithms under different problem sizes. Furthermore, a comparison between the corresponding simulation results is performed in order to validate both the problem formulation and the developed fog computing simulator. Afterwards, a comparison between these algorithms is performed by considering static and location aware scenarios. To this end, a single static scenario is considered where some mobile devices are introduced. Although being mobile devices, its mobility mechanisms are deactivated. As the controller obeys to the rule to connect mobile devices to their closest fog or cloud node, generating initial random positions while running multiple times the same scenario allows to perform a comparison between these algorithms. This is performed by comparing their average solutions and consequent simulation results under different settings. Moreover, the impact of the relative tolerances is studied in order to mitigate the objective priority impositions. Finally, the mobile environments are tested. The latter allows to verify the impact, limitations and benefits of the migrations.

4.1 Simulation Parameters

The main simulation parameters, fixed in all the simulations performed in this chapter, are listed in Table 4.1. As already mentioned, monetary values are meaningless when considering the system provider perspective. As these parameters are solely used when assuming the broker perspective, they are set to zero and are not considered in the simulation results. Also, as already discussed, in Section 3.2, both latency and bandwidth values for mobile communications are constant in the whole 2D plane. In this case, envisioning the 4G/5G communications, it is considered that each cellular link is capable of transmitting 400 Mb/s and being characterized by zero latency. Moreover, regarding the transmission power, it is set to 23 dBm (a well known value) which corresponds to the maximum transmission power of User Equipment (UE) for LTE communications [91].

As already discussed, the optimization is performed in a priority-based fashion. Therefore, each
Table 4.1: List of fixed simulation parameters.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f^\text{Mem})</td>
<td>Memory capacity of each node</td>
<td>1GB</td>
</tr>
<tr>
<td>(f^\text{Strg})</td>
<td>Storage capacity of each node</td>
<td>4GB</td>
</tr>
<tr>
<td>(f^\text{Tx})</td>
<td>Cellular network transmitter power of each node</td>
<td>23dBm</td>
</tr>
</tbody>
</table>

**Communication**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E^L)</td>
<td>Link latency for each mobile network link</td>
<td>0ms</td>
</tr>
<tr>
<td>(E^Bw)</td>
<td>Link bandwidth for each mobile network link</td>
<td>400Mb/s</td>
</tr>
</tbody>
</table>

**Controller**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha^p)</td>
<td>Percentage of processing resources used for useful work</td>
<td>0.95</td>
</tr>
<tr>
<td>(\alpha^m)</td>
<td>Percentage of memory resources used for useful work</td>
<td>0.95</td>
</tr>
<tr>
<td>(\alpha^s)</td>
<td>Percentage of storage resources used for useful work</td>
<td>0.95</td>
</tr>
<tr>
<td>(\alpha^b)</td>
<td>Percentage of bandwidth resources used for useful work</td>
<td>0.8</td>
</tr>
<tr>
<td>(t^\text{Boot})</td>
<td>Constant VM setup time</td>
<td>10s</td>
</tr>
</tbody>
</table>

The objective is assigned a different priority. As there are too many possibilities, in this chapter, only the following four different combinations are only considered:

- **Priority set 1**: \(\text{QoS} = 5, \text{Power} = 4, \text{Processing} = 3, \text{Bandwidth} = 2, \text{Migration} = 1\);
- **Priority set 2**: \(\text{QoS} = 4, \text{Power} = 5, \text{Processing} = 3, \text{Bandwidth} = 2, \text{Migration} = 1\);
- **Priority set 3**: \(\text{QoS} = 4, \text{Power} = 3, \text{Processing} = 5, \text{Bandwidth} = 2, \text{Migration} = 1\);
- **Priority set 4**: \(\text{QoS} = 4, \text{Power} = 3, \text{Processing} = 2, \text{Bandwidth} = 5, \text{Migration} = 1\).

Moreover, the priority-based optimization also implements relative cost tolerances. Similarly to the above mentioned, this chapter only considers the following five different combinations (in which the Tolerance set 0 is the one implemented when omitted):

- **Tolerance set 0**: \(\text{QoS} = 0, \text{Power} = 0, \text{Processing} = 0, \text{Bandwidth} = 0, \text{Migration} = 0\);
- **Tolerance set 1**: \(\text{QoS} = 0.33, \text{Power} = 0, \text{Processing} = 0, \text{Bandwidth} = 0, \text{Migration} = 0\);
- **Tolerance set 2**: \(\text{QoS} = 0.5, \text{Power} = 0, \text{Processing} = 0, \text{Bandwidth} = 0, \text{Migration} = 0\);
- **Tolerance set 3**: \(\text{QoS} = 0.5, \text{Power} = 0.33, \text{Processing} = 0, \text{Bandwidth} = 0, \text{Migration} = 0\);
- **Tolerance set 4**: \(\text{QoS} = 0.5, \text{Power} = 0.5, \text{Processing} = 0, \text{Bandwidth} = 0, \text{Migration} = 0\).

In this chapter, both the GA and the RA are tested using two different configurations. These are defined as presented in Table 4.2.

Similarly to the work performed in [75], the proposed system performance is tested using two different types of applications, each of which, composed of real-time interactions. These are described as follows:

- **EEG Tractor Beam Game**. Is a human-vs-human game that involves augmented brain-computer interaction [92]. In order to play the EEG Tractor Beam game, each player needs to wear an EEG headset that is connected to his smartphone. Then, the application performs real-time processing of the EEG signals sensed and calculates the brain state of the user. On the display, the game
Table 4.2: List of the optimization algorithm parameters used in this section.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Config. 1</th>
<th>Config. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Genetic Algorithm</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>POPULATION SIZE</strong></td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td><strong>POPULATION SIZE</strong></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td><strong>MAX_ITER</strong></td>
<td>100000</td>
<td>100000</td>
</tr>
<tr>
<td><strong>MAX ITER</strong></td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td><strong>MAX_ITER</strong></td>
<td>10</td>
<td>25</td>
</tr>
<tr>
<td><strong>MAX_CONVERGENCE</strong></td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td><strong>Random Algorithm</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MAX_ITER</strong></td>
<td>1000000</td>
<td>1000000</td>
</tr>
<tr>
<td><strong>MAX_CONVERGENCE</strong></td>
<td>2000</td>
<td>13000</td>
</tr>
</tbody>
</table>

Figure 4.1: DAG of the EEG Tractor Beam Game.

shows all the players on a ring surrounding a target object. Each player can exert an attractive force onto the target in proportion to his level of concentration. In order to win the game, a player should try to pull the target toward herself/himself by exercising concentration while depriving other players of their chances to grab the target. This application can be modeled as a set of modules, forming a DAG as depicted in Figure 4.1. As illustrated, the EEG Tractor Beam Game have three major modules which perform processing - Client, Concentration Calculator and Coordinator. Table 4.3 presents the application edges characteristics [75], and Table 4.4 provides the data dependencies between edges under the form of fractional selectivity. Meanwhile, the application loop is given under the following path: EEG → Client → Concentration Calculator → Client → Display.

- **Intelligent Surveillance System.** This application aims to coordinate multiple cameras with dif-

Table 4.3: Application edges of the EEG Tractor Beam Game.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Processing [(MIT)]</th>
<th>Data [(B)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEG</td>
<td>300</td>
<td>500</td>
</tr>
<tr>
<td>SENSOR</td>
<td>350</td>
<td>500</td>
</tr>
<tr>
<td>PLAYER_GAME_STATE</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>CONCENTRATION</td>
<td>1.4</td>
<td>500</td>
</tr>
<tr>
<td>GLOBAL_GAME_STATE</td>
<td>2.8</td>
<td>1000</td>
</tr>
<tr>
<td>GLOBAL_STATE_UPDATE</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>SELF_STATE_UPDATE</td>
<td>0</td>
<td>500</td>
</tr>
</tbody>
</table>
ferent Fields Of View (FOVs) to surveil a given area. Coordination between cameras involves coordinated tuning of Pan–Tilt–Zoom (PTZ) parameters so that the best view of the area can be obtained. Furthermore, the system alerts the user in case of irregular events - which may demand attention of the security authorities [75]. This application is also modeled as a set of modules, forming a DAG as depicted in Figure 4.2. Table 4.5 presents the application edge characteristics [75] and Table 4.6 provides the data dependencies between edges under the form of fractional selectivity. In this application, there are two application loops: Motion Detection → Object Detection → Object Tracker, and Object Tracker → PTZ Control.

### 4.2 Behaviour and Performance of the Proposed System

In this section, the performance of the proposed system is analysed. To this end, Section 4.2.1 performs a detailed analysis of a simple, static and location unaware scenarios in order to validate both the proposed problem formulation and the developed fog computing simulator, as well to verify the different proposed decision-making algorithm performances under different problem sizes. Then, Section 4.2.2 compares the average performance of each algorithm while implementing more complex, static and

<table>
<thead>
<tr>
<th>Module</th>
<th>Input</th>
<th>Output</th>
<th>Fract. selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>EEG SENSOR</td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Client</td>
<td>CONCENTRATION</td>
<td>SELF_STATE_UPDATE</td>
<td>1.0</td>
</tr>
<tr>
<td>Concentration Calculator</td>
<td>SENSOR</td>
<td>CONCENTRATION</td>
<td>1.0</td>
</tr>
<tr>
<td>Client</td>
<td>GLOBAL_GAME_STATE</td>
<td>GLOBAL_STATE_UPDATE</td>
<td>1.0</td>
</tr>
</tbody>
</table>

![DAG](image)”}

**Table 4.5: Application edges of the Intelligent Surveillance System.**

<table>
<thead>
<tr>
<th>Edge</th>
<th>Processing [M]</th>
<th>Data [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAMERA</td>
<td>100</td>
<td>2000</td>
</tr>
<tr>
<td>MOTION_VIDEO_STREAM</td>
<td>200</td>
<td>2000</td>
</tr>
<tr>
<td>DETECTED_OBJECT</td>
<td>50</td>
<td>2000</td>
</tr>
<tr>
<td>OBJECT_LOCATION</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>PTZ_PARAMS</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>
Table 4.6: Application dependencies of the Intelligent Surveillance System.

<table>
<thead>
<tr>
<th>Module</th>
<th>Input</th>
<th>Output</th>
<th>Fract. selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Detection</td>
<td>CAMERA</td>
<td>MOTION_VIDEO_STREAM</td>
<td>1.0</td>
</tr>
<tr>
<td>Object Detection</td>
<td>MOTION_VIDEO_STREAM</td>
<td>OBJECT_LOCATION</td>
<td>1.0</td>
</tr>
<tr>
<td>Object Detection</td>
<td>MOTION_VIDEO_STREAM</td>
<td>DETECTED_OBJECT</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Figure 4.3: First scenario: a static fog environment composed by two interconnected fog, one cloud and one client nodes.

location aware environments. This is followed by Section 4.2.3 which studies the usage of the relative tolerances in order to mitigate the cost priorities impositions. Finally, Section 4.2.4 studies the impact, limitations and benefits of migrating VMs in mobile environments.

4.2.1 Simulator, Architecture, and Problem formulation Validation

In order to validate the developed system functionality, this section only considers static environments without location awareness. This means that all nodes are static and their connections are well-defined. In this way, the controller does not seek to connect mobile nodes to their closest fog or cloud node. In order to compare the system performance using different decision-making algorithms, it is required to compare their own performance and solutions found, under the presence of different cost priorities, and if these are effectively reflected in the simulation results. This section presents and compares three different scenarios with different numbers of nodes, network links, applications, edges and modules. In these tests, all costs assume zero relative tolerances, each algorithm is only executed once, and the simulation time is set to \( 10 \) s. The first test scenario is depicted in Figure 4.3. As it can be seen, the system is composed of four fog nodes and one client node, resulting in \( N = 5 \). These are connected using \( E = 10 \) edges. Note that the F2 node is considered to be connected to a green power source. The client aims to deploy the EEG Tractor Beam Game (with \( M = 5, Z = 6 \)). Although being a human-vs-human game, for the test purposes it is assumed to be a player-vs-machine game. Therefore, the sizes of the problem variables are given as follows: \( P \rightarrow 5 \times 5 \) and \( R \rightarrow 6 \times 10 \). In this test, the EEG module
generates a new tuple every $5ms$ and its loop deadline is set to $25ms$. The different algorithm iteration-value plots for the priority set 1, priority set 2, priority set 3, and priority set 4 are presented in Figures C.1, C.2, C.3, and C.4, respectively. As all algorithms were able to find the optimal solution, Table 4.7 only presents the values corresponding to the CPLEX optimizer solution. Nonetheless, the average algorithm execution times (regarding the four performed tests) for the CPLEX optimizer, GA-Config. 1, BFA, and RA-Config. 1 were $367ms$, $226ms$, $24ms$, $307ms$, respectively. 

Bearing in mind the sensor periodicity (i.e., $5ms$) and the required average processing capacity, computed using the formulation described in Equation 3.3, the Client, Concentration Calculator, and Coordinator modules require $60.308MIPS$, $70.000MIPS$, and $1.000MIPS$, respectively. The Client module, being of type client, needs to be always deployed inside the client node. As the controller considers $\alpha_p = 0.95$, the Client node has no available space to support any other module. From table 4.7, it is possible to verify that the Client node has always a fixed occupation ($\approx 94.04\%$) for all priority sets, which corresponds to approximately $\frac{60.308 \times 100}{64.900} \approx 94.23\%$. Regarding the priority set 1, it is possible to verify that the Proxy hosts the Concentration Calculator module ($\frac{70.000 \times 100}{1.000.000} = 7\%$) in order to meet the defined loop deadline ($25ms$) in the worst case scenario, given by the Equations 3.8 and 3.9. Note that, the simulated loop values are below the computed $24.75ms$. Meanwhile, as the Coordinator module is not defined in the application loop, it can be deployed in any node without regard its impact onto the user application. As the second highest priority defined in the priority set 1 is the power consumption, it is hosted by the F2 node, which was null power consumption. Still in priority set 1, it is possible to validate the power consumption for the Proxy node which corresponds to approximately $(83,4333 + (107,339 - 83,4333) \times 0.07) \times 10 \approx 851.067J$, where $10s$ corresponds to the simulation time. For sake of simplicity (due to high number of application edges and network links), the network link bandwidth utilization is
Cloud
Proxy
F1 F2
Client1
fMips  = 44.800.000 MIPS
fbPw  = 1.648 W
fiPw  = 1.332 W
fMips  = 1.000.000 MIPS
fbPw  = 107,339 W
fiPw  = 83,433 W
fMips  = 75.000 MIPS
fbPw  = 50 W
fiPw  = 38 W
fMips  = 1.000.000 MIPS
fbPw  = 0 W
fiPw  = 0 W
fMips  = 64.000 MIPS
fMips  = 75.000 MIPS
fMips  = 75.000 MIPS
fMips  = 1.000 MIPS

**Figure 4.4:** Second scenario: a static fog environment composed by three interconnected fog, one cloud and two client nodes.

abstracted from this explanation. Nonetheless, it can be verified that both total CPU and total NW occupation reflect approximately the values computed by the algorithm. For instance, in *priority set 1*, $0.08771 \times \alpha^p \simeq 0.0833245 \simeq 8.2956\%$, and $0.04125 \times \alpha^b = 0.033 \simeq 3.2925\%$. It is also of interest to notice that the *priority set 4* is also able to always accomplish the loop deadline even though the algorithm has estimated the opposite. Also, the *priority set 2* allows for achieving the lowest total energy consumption, *priority set 3* the lowest total CPU occupation, and the *priority set 4* the lowest total NW occupation. Note that the total values respect only to the system provider resources. In this way, as both the algorithm estimates and the simulation results meet the expected values, and these are optimized according to the defined priorities, it is concluded the validity of the proposed model and problem formulation.

For sake of comparison of the algorithm execution times and its solutions under different problem sizes, two additional scenarios were tested. These are depicted in Figures 4.4, and 4.5, respectively. For the second test scenario, the previous *Client* corresponds to the *Client1* (with the same characteristics), and *Client2* is added while aiming to deploy the application Intelligent Surveillance System. Therefore, the sizes of the problem variables are given as follows: $P \rightarrow 6 \times 9$ and $R \rightarrow 11 \times 12$. In this test, the *Camera* module of the user generates a new tuple every 5ms and its two loop deadlines are set to 10ms. In this case, the average algorithm execution times for the CPLEX optimizer, GA-Config. 1, BFA, and RA-Config. 1 were 1250ms, 589ms, 2266ms, 715ms, respectively. Note that in contrast to the CPLEX optimizer and the BFA, in this scenario, both the GA and the RA were not able to find the optimal solution. For the third considered scenario, *Client1* and *Client2* remain with the same characteristics. *F3* is added to the physical infrastructure, as well as the *Client3*. The latter shares the same characteristics of the *Client2*. Therefore, the sizes of the problem variables are given as follows: $P \rightarrow 8 \times 15$ and $R \rightarrow 16 \times 16$. In this test, the BFA was not able to run in feasible time. Meanwhile, the GA and the RA, similarly to the above example, were not able to find the optimal solutions. Nonetheless, the average algorithm execution times for the CPLEX optimizer, GA-Config. 1, and RA-Config. 1 were 8666ms,
In order to compare the performance of the different proposed algorithms, a static environment with location awareness was implemented. In this environment, the physical infrastructure of the system provider is composed of a Cloud node, a Proxy node, and four different Fog nodes. Note that these node names are an analogy to the architecture depicted in Figure 4.3. Regarding the communications, the Proxy is connected to the Cloud, as well as to the different static Fog nodes. In order to introduce more diversity to the system, one of the four Fog nodes is characterized by null power consumption and another is mobile. Meanwhile, the system is also composed of four similar mobile Client nodes, each of which aiming to deploy the Intelligent Surveillance System application. Therefore, the sizes of the problem variables for the current scenario are given as follows: \( P \to 10 \times 24 \) and \( R \to 20 \times 18 \).

In this test, the BFA is not able to run. In this way, this section only compares the performances of the CPLEX optimizer, GA, and RA. This is performed through successive simulations of the considered scenario while implementing each decision-making algorithm. For that purpose, the simulation area is defined with a square side size of 2,000 m. All nodes are then placed in a random physical position within the simulation area. Then, the controller is responsible to connect each mobile node to its closest fog or cloud node. Therefore, in each simulation, the controller needs to optimize the system performance under different characteristics. Note that, although being mobile nodes, these remain in their initial physical positions.

This test is performed 10 times for each algorithm, always using the priority set 1. The results are presented in Table 4.8. In this table it is possible to verify that the CPLEX optimizer, while finding the
optimal solution, was always able to accomplish the defined loop deadlines. Note that, in this scenario,
it is always possible to accomplish the defined deadlines, even though it is possible to verify that the
standard deviation relative to the CPLEX execution time is very high compared to its average time. This
behaviour occurs due to the fact that with different randomly selected physical positions of the nodes
(which creates different network links), it can be more difficult to find the optimal solution.

Still in the same table, the loop deviation parameter corresponds to the average loop delay of the
loops that the algorithm has estimated to accomplish. It is computed by the sum of all differences
between the corresponding defined deadline and the simulated latency of the ensured application loops
over the sum of all completed loops relative to the same ensured application loops. It was observed
that, when using the CPLEX optimizer, a few application loop deadlines were not accomplished. This
behaviour occurs due to the fact that the algorithms, while seeking for optimizing all cost, prefer to set
the application loops worst case scenario value closer and below to the defined deadline as long as
it brings advantages to the other objectives. As this value only respects to the average value and the
CPLEX is the only one able to achieve the optimal solutions, a few loops were not satisfied while using
it. Regarding the formulation presented in Equation 3.8, it is considered that, for a given time instant, the
CPU of a given node is free and it receives all possible tuples for its hosted modules at the same time. It
is also considered that, the dependency which the latency is being computed is the last to be executed
(worst case). As also shown, the processing requirements of each module are computed based on their
dependency periodicities and probabilities. Therefore, reducing the dependency probabilities results in
a reduction of the processing requirements. In this way, if a given node is nearly using its full capacity,
some loop violations may occur. While following a given probability, tuples may arrive to its destination
module with an higher tuple rate than the expected average value. For instance, if the probability of a
given tuple arrive to its destination module is $0.5$ every $10\text{ms}$, there will occur cases where 2 tuples arrive
in $20\text{ms}$. However, the problem formulation considers that every $20\text{ms}$, the module only receives 1 tuple.
In such cases, an higher processing delay is experienced, which may translate into a loop violation.
Therefore, only the average value of the application loops is ensured.

Moreover, although only two performances are presented (due to space constraints), it can be verified
that the solutions found by the GA can be further enhanced by tuning its parameters (refer to Table 4.2).
As its new solutions are generated based on its previously achieved knowledge through successive
generations, increasing the number of individuals in each population allows it to achieve more diversity
of solutions, which, in turn, results in achieving better solutions. Moreover, increase its convergence
stop condition will also mitigate its premature convergence allowing to achieve better results. In this
tests, RA has achieved the worst performances. Its parameters for the Config. 2 were selected in order
to achieve similar execution times compared to the GA-Config. 2. However, as it can be verified, its
improvements compared to the RA-Config. 1 are negligible. This behaviour was expected due to the
fact that, contrary to the GA, it does not use the previously found solutions in the calculation of the new
ones. From this table, once again, it can be verified that the average simulation results mach the average
algorithm results. Note that this behavior is reflected in every parameter of every algorithm except for
the network usage in the RA. As presented in Algorithm 5, the random routing path generation does
Table 4.8: Performance comparison of the different proposed optimization algorithms.

<table>
<thead>
<tr>
<th>Algorithm Estimates</th>
<th>CPLEX</th>
<th>GA-Config. 1</th>
<th>GA-Config. 2</th>
<th>RA-Config. 1</th>
<th>RA-Config. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exe. time [ms]</td>
<td>57010,5</td>
<td>10473,1</td>
<td>51647</td>
<td>7123,1</td>
<td>51717,8</td>
</tr>
<tr>
<td>Exe. time s.d. [ms]</td>
<td>29516,54</td>
<td>2602,44</td>
<td>11444,88</td>
<td>2144,93</td>
<td>37724,45</td>
</tr>
<tr>
<td>QoS value</td>
<td>0,4</td>
<td>1,2</td>
<td>0,4</td>
<td>2,4</td>
<td>2,1</td>
</tr>
<tr>
<td>Power value</td>
<td>0,49882</td>
<td>5,83741</td>
<td>11,1925</td>
<td>9,0536</td>
<td>10,5596</td>
</tr>
<tr>
<td>Processing value</td>
<td>0,23222</td>
<td>0,79535</td>
<td>0,80216</td>
<td>0,87309</td>
<td>0,927475</td>
</tr>
<tr>
<td>Bandwidth value</td>
<td>0,1515</td>
<td>0,28923</td>
<td>0,17707</td>
<td>1,08821</td>
<td>1,493319</td>
</tr>
<tr>
<td>Simulation Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop deviation [ms]</td>
<td>0,08758</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total energy [J]</td>
<td>16662,067</td>
<td>16712,73</td>
<td>16763,59</td>
<td>16742,69</td>
<td>16757,32</td>
</tr>
<tr>
<td>Total CPU occ. [%]</td>
<td>21,91679</td>
<td>74,670418</td>
<td>75,69641</td>
<td>81,93536</td>
<td>87,07744</td>
</tr>
<tr>
<td>Total NW occ. [%]</td>
<td>12,11133</td>
<td>23,15415</td>
<td>14,15426</td>
<td>36,91411</td>
<td>8,94607</td>
</tr>
</tbody>
</table>

Table 4.9: Impact comparison of different tolerance sets using the GA with the Config. 2 parameters.

<table>
<thead>
<tr>
<th>Tol. set 0</th>
<th>Tol. set 1</th>
<th>Tol. set 2</th>
<th>Tol. set 3</th>
<th>Tol. set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS value</td>
<td>0,4</td>
<td>0,3</td>
<td>2,1</td>
<td>2,2</td>
</tr>
<tr>
<td>Power value</td>
<td>11,1925</td>
<td>4,01428</td>
<td>1,13023</td>
<td>1,89349</td>
</tr>
<tr>
<td>Processing value</td>
<td>0,80216</td>
<td>0,43944</td>
<td>0,51758</td>
<td>0,31808</td>
</tr>
<tr>
<td>Bandwidth value</td>
<td>0,17707</td>
<td>0,06783</td>
<td>0,65662</td>
<td>0,47124</td>
</tr>
<tr>
<td>Simulation Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total energy [J]</td>
<td>16763,59</td>
<td>16695,44</td>
<td>16668,06</td>
<td>16675,29</td>
</tr>
<tr>
<td>Total CPU occ. [%]</td>
<td>75,69641</td>
<td>41,40978</td>
<td>48,6356</td>
<td>29,98647</td>
</tr>
<tr>
<td>Total NW occ. [%]</td>
<td>14,15426</td>
<td>4,50257</td>
<td>52,46875</td>
<td>37,60341</td>
</tr>
</tbody>
</table>

not verify if there are repeated hops. Therefore, some solutions may oversize the required bandwidth. While in the GA this is hidden due to its gathered knowledge when defining new solutions, in the RA, for relatively large scenarios, it will probably find oversized solutions regarding the bandwidth objective.

### 4.2.3 Study of the Impact of the Relative Tolerances

In this section, the impact of the relative tolerances is studied. As already mentioned, it is important to provide flexibility to the system in order to mitigate the impact of the imposed objective priorities. To this end, it is used the scenario defined in Section 4.2.2. The results are presented in the Table 4.9. Note that each test is also executed 10 times, using the GA with the Config. 2 parameters and the priority set 1.

As it can be verified, when using the Tol. set 1, its impact on the QoS is negligible (in this case it was verified a slight improvement). However, its impact on the power objective is significant. Since a QoS objective tolerance of 33.3% is applied, it is able to focus more on the power consumption, and, consequently achieve better results regarding this metric, which is also reflected in the simulations results. Regarding the Tol. set 2, the impact in the QoS objective is more noticeable. In this case, the GA prefers to give QoS assurance to only 50% of the total number of applications aiming to achieve a better power consumption performance, which is noticeable in both the algorithm results and the simulation itself. With respect to the Tol. set 3, the 50% QoS tolerance is maintained and a 33.3% tolerance is added to the power objective. While doing so, the impact on the QoS objective is negligible and the
power consumption increases for the sake of reducing the processing objective value, which can be verified both in the algorithm and the simulation results. Finally, regarding the Tol. set 4, where the QoS tolerance is maintained at 50% and the tolerance for the power objective is increased to 50%, the increase both in the QoS and the Power values in order to reduce the processing value is noticeable. This behaviour (i.e., the impact on the QoS) is verified due to the fact that the average rate of new better solutions is reduced. Therefore, its solutions are less optimal regarding the QoS. Nonetheless, as expected, it was able to further reduce the processing cost, which is also reflected in the simulation results.

Note that this test could be implemented using any of the proposed algorithms, any possible combination of priorities, and any possible combination of relative tolerances. However, due to space constraints, this test is restricted to the above mentioned sets. Nonetheless, it serves of proof of concept, showing that it is possible to reduce the impact of the defined objective priorities, allowing more flexibility to the system.

### 4.2.4 Study of the Impact of the Migrations

In this section it is studied the impact, limitations and benefits of the use of migrations while considering mobile environments. To this end, it is tested the environment depicted in Figure 4.6. In this environment, the Cloud, Proxy, and F1 are static and interconnected nodes. Then, the system is composed of three mobile nodes (i.e., F2, Client1, Client2). F2 is periodically connected to the Cloud, Proxy, and F1 nodes. Meanwhile, the Client1 periodically changes its connection between the Cloud and F1 nodes. Finally, the Client2 periodically changes its connection between the Cloud and Proxy nodes. In this scenario, both users aim to deploy the Intelligent Surveillance System in which the application loop Motion Detection → Object Detection → Object Tracker is defined with a deadline of 15 ms, and Object Tracker → PTZ Control is defined with a deadline of 10 ms. In this case, the system is tested using the CPLEX optimizer, in order to optimize the defined scenario in a deterministic manner. Note all nodes are characterized with the same velocities, each of which following its corresponding deterministic path.
Also, in this scenario the simulation time is changed to 300s. The simulation results are presented in Table 4.10. Set 1 implements the never migrate method, and Set 2, Set 3, and Set 4 obey to the priority set 1, where VM or container sizes (Memory + Storage) are characterized of 5MB, 50MB, and 500MB, respectively. Regarding the values referring to the velocity of 5km/h, it is possible to verify that, when implementing the Set 1, the Client 1 has almost all loops corresponding to its second loop violated. This only occurs in Client 1 due to the fact that is the only client node to suffer an handover (due to the initial physical positions). Meanwhile, when implementing the Set 2 due to the small VM size, its application in terms of the number of accomplished loops is negligibly affected. On the contrary, its number of violated loops is reduced (in this case is zero). Nonetheless, it is possible to verify that the Client 2, due to its module placements, suffers some loop violations during the migration of VMs of the Client 1. This behaviour increases when increasing the size of the VMs. In the same way, the number of tuples lost also increases. Note that in Set 4, there are 4 migrations instead of 3 because, in each machine, \( \alpha \times f^{Mem} \) only allows to support 3 modules of 500MB. Looking through the values referring to the velocity of 50km/h, this behaviour is more evident. For instance, the number of loops not accomplished while using the Set 1, increases substantially for both clients due to the occurrence of handovers (which connect the clients to more network distant nodes compared to the ones hosting its modules). Moreover, when using the Set 1 through Set 4, the number of violated loops is mitigated. However, the number of completed loops is increasingly reduced due to the number of tuples lost during migrations. In this case, as expected, the number of migrations is reduced as the VM sizes increase. This is explained due to the fact that, as migrations take longer, when handovers occur, it is more probable that the migration was already being executed. Therefore, even though the controller decides that the module currently being migrated should be migrated to a different destination, the number of migrations is only incremented once. Finally, looking through the values referring to the velocity of 120km/h, it is possible to verify that this behaviour is even more noticeable. When implementing Set 4, the number of completed loops regarding the Client 1 is extremely reduced, and Client 2 not even completes a single loop (due to high nodes mobility). Even so, it is clear to see that migrations can be used in order to benefit both the clients and the system provider objectives. In this case, the defined deadline for migrations was set to a sufficient high value so that the controller would always migrate the VMs in order to reduce the QoS objective value. However, as the deadlines are defined by the user itself, the number of tuples lost is not relevant. Moreover, as both F1 and F2 nodes were defined with a null power consumption (e.g., when deployed inside a bus or connected to a green power source), the use of migrations has allowed to reduce the total system power consumption (refer to Table 4.10), provided that the QoS cost is already minimized.

4.3 Summary

In this chapter, the results of the tests regarding the proposed architecture, problem formulation, optimization algorithms, and the developed fog simulator have been presented. From Table 2.2, it is possible to verify that all the proposed features, whose integration is currently lacking in the literature, were im-
Table 4.10: Migration impact under different velocities and VM sizes.

<table>
<thead>
<tr>
<th></th>
<th>Velocity = 5km/h</th>
<th>Velocity = 50km/h</th>
<th>Velocity = 120km/h</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set 1</td>
<td>Set 2</td>
<td>Set 3</td>
</tr>
<tr>
<td># Completed C1 L1</td>
<td>59979</td>
<td>57893</td>
<td>57199</td>
</tr>
<tr>
<td># Violated C1 L1</td>
<td>2996</td>
<td>2256</td>
<td>2065</td>
</tr>
<tr>
<td># Completed C1 L2</td>
<td>2619</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td># Violated C1 L2</td>
<td>2999</td>
<td>2246</td>
<td>1986</td>
</tr>
<tr>
<td># Completed C2 L1</td>
<td>59979</td>
<td>57893</td>
<td>57199</td>
</tr>
<tr>
<td># Violated C2 L1</td>
<td>2996</td>
<td>2256</td>
<td>2065</td>
</tr>
<tr>
<td># Completed C2 L2</td>
<td>2619</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td># Violated C2 L2</td>
<td>2999</td>
<td>2246</td>
<td>1986</td>
</tr>
<tr>
<td>Total NW occ. [%]</td>
<td>8.4628</td>
<td>15.7501</td>
<td>18.1545</td>
</tr>
<tr>
<td>Total Energy [J]</td>
<td>424757</td>
<td>424655</td>
<td>424655</td>
</tr>
<tr>
<td># Packet success</td>
<td>366074</td>
<td>359465</td>
<td>357384</td>
</tr>
<tr>
<td># Packet drop</td>
<td>0</td>
<td>2103</td>
<td>2837</td>
</tr>
<tr>
<td># Handover</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># Migration</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

implemented and studied. The proposed fog system is flexible enough to benefit from any node deployed between the end users and the cloud in order to enhance users perceived QoS. The users can deploy any type of application into the system, where applications structured as a set of modules using the DDF programming model can be further enhanced while being deployed, in multiple nodes, rather than running in a single node. In the developed simulator, contrary to the original iFogSim, these can be deployed in any fog node due to the implemented directed graph-based architecture.

Regarding the optimization of the system performance, it has been shown that the BF is able to find the optimal solutions in a short period of time when considering small scenarios. However, due to its nature it is not able to solve the problems for medium-large scenarios. The CPLEX optimizer is also able to find the optimal solutions. However, depending on the system characteristics, it can be characterized of high execution times. Meanwhile, the GA and the RA are able to find near-optimal solutions in a reduced execution time. Nonetheless, only the GA is able to improve the solutions when tuning its parameters. Also, even though the optimization is performed in a priority-based fashion, the system can achieve more flexibility using relative tolerance in the objectives. Finally, it was shown that the migrations can be used in order to further enhance the system provider objectives. Nonetheless, the latter still has performance limitations when considering higher VMs sizes and highly mobile environments.
Chapter 5

Conclusions

In this thesis, a novel fog computing system architecture and optimization problem formulation have been proposed and developed in a computer simulation tool. The proposed architecture allows the mobility of IoT and fog nodes while seeking to preserve the user QoS through migration mechanisms. The proposed problem formulation allows to optimize the system performance from a system provider perspective. Nevertheless, it is worth noting that it is ready to be configured to optimize the system from a computing resource broker perspective, if such a business model is considered. The proposed framework was tested through computer simulation under different decision-making algorithms. The developed simulation tool significantly extended the existing iFogSim simulator, making it more flexible and able to support more complex modelling, with emphasis on client and resource mobility.

Fog computing has still more than one definition in the literature. It is consensual that is structured as a network of interconnected nodes with processing capabilities, using a wide range of communication technologies which stand between client endpoints (e.g., sensors, actuators, mobile user terminals) and the cloud. These nodes intend to offer their computing power at the service of delay sensitive client applications, in order to provide responses faster than the cloud. Fog computing has a generic nature associated with the fact that it does not rely on any specific designed and purposely built technology. Therefore, the proposed architecture, in contrast to most previously proposed ones, is generic regarding all nodes, communications and applications. Additionally, bearing in mind dynamic environments, the proposed architecture also assumes the presence of mobile fog and client nodes. Migration of services between nodes, is supported and can be explored in order to further enhance the QoS perceived by the users in mobile environments. Provided that constraints (e.g., delay bounds) are fulfilled, its up to the orchestrator, based on its predefined objectives, to decide which node should host and execute each user service. This decision-making and its impact onto the system performance were tested using several optimization algorithms including the CPLEX optimizer, brute force, random search and genetic algorithm. The performance evaluation of the proposed system is achieved through simulation of different scenarios in the developed fog computing simulator, which further extends the iFogSim simulator capabilities. The performance impact of each component of the system was assessed through some relevant QoS metrics regarding both the client and the system provider objectives. As a preliminary
step, a set of simulations using different static scenarios has been assessed without the influence of mobility and location awareness mechanisms. The expected behavior of the developed simulator, the problem formulation and the different decision-making algorithms was thus validated prior to the more complex scenarios. This also allowed a first comparison between the algorithms and identification of their limitations. In particular the brute force algorithm, as expected, has to consider a prohibitively large number of candidate solutions when increasing the problem size, which is characterized by the number of nodes, network links, application dependencies and modules. Also, the performances of both the random and the genetic algorithms are also degraded by the increase of the problem size. Then, the performance of more complex scenarios using location awareness has been assessed without the influence of mobility. On the one hand, it is shown that, in contrast to the purely random algorithm, the genetic algorithm is able to be further enhanced by tuning its configuration parameters due to its gathered knowledge through successive generations. It is also verified that the CPLEX optimizer execution time, while seeking for the optimal solution, is highly dependent on the problem search space. On the other hand, it was demonstrated that even though the problem is being optimized in a priority-based fashion, the usage of relative objective tolerances can be implemented in the interest of providing more flexibility to the system. Finally, the system performance is assessed in a simple and deterministic mobile environment. The latter has allowed to verify the achievable benefits of implementing mobility-triggered migrations of the VMs. These benefits concern both the perceived QoS by the users and the system provider objectives. Nonetheless, it was also verified that this mobility management mechanism has its limitations, which are most notorious when the VM sizes increase and in the presence of highly dynamic environments. In such environments, partition into independently managed fog colonies is a promising technique that should be explored in extensions of this work.

5.1 Future Work

There are some additional points that can be explored to continue and possibly upgrade what was accomplished in the present work:

- The movement of the nodes has been modeled with simple mobility patterns which do not realistically portray the scenarios where the fog computing is intended to be implemented (e.g., smart cities). More realistic and sophisticated mobility patterns can be implemented based on real data sets;

- The formulation of the worst case scenario latency assumes regular traffic without bursts, which may translate into some application loop violations in practice. In a more sophisticated system, feedback mechanisms should be implemented, allowing to calculate statistics on real traffic patterns, which could be taken into account together with the already computed average. For instance, in this case, the upper limit of the confidence interval could be used as a more conservative estimate;

- This work only considers the presence of 4G/5G cellular communications with full coverage. This
can be further extended by implementing other mobile communication technologies (e.g., Wi-Fi).
The implementation of *dead zones* can also be explored;

- When migrations or handovers occur, some tuples may be lost. A more sophisticated system could implement efficient rerouting protocols in order to mitigate the impact onto the users when these occur. Moreover, the migrations impact can be even further mitigated. First, by implementing live migration mechanisms, which reduces the total down time service. Then, by effectively allocating bandwidth resources for migration purposes in the simulator, the impact of migrations in the QoS of other users could be mitigated. A mobility-aware proactive-based approach optimization, should further reduce the impact of migrations;

- The implemented architecture considers that the problem size remains constant. In more realistic environments, clients could dynamically enter or leave the system. Moreover, when the desired QoS is not ensured by the optimization algorithm, non-satisfiable clients could be removed from the system in order to improve the service of satisfiable clients. In this way, the QoS cost could be implemented as a constraint instead of an optimization objective;

- The implemented architecture still has some limitations due to the high complexity problem formulation. Instead of following a centralized approach, it could be explored the concept of fog colonies. Each colony could be delimited by a graph partitioning technique, number of hops, physical distance, etc. In this way, each colony orchestrator could implement the proposed problem formation. Then, the global optimization could be implemented using a distributed approach or even using a hierarchy of orchestrators in order to simultaneously increase scalability and resource efficiency;

- The system performance has been tested from the system provider perspective. Using the flexible proposed problem formulation, a broker perspective can also be studied in the future.
Bibliography


Appendix A

Optimization Algorithms

Algorithm 1 presents the recursive approach adopted for the implementation of the brute force algorithm. The pseudocode presented in Algorithm 2 describes the implemented procedure in order to compare two different solutions. Algorithm 3 depicts the execution of the random search algorithm. The procedure presented in Algorithm 4 is implemented for the generation of a given pseudorandom placement matrix. Meanwhile, the procedure presented in Algorithm 5 is implemented for the generation of a given pseudorandom routing matrix. Algorithms 6 through 9 describe the procedures adopted for implementing the two-step genetic algorithm.

Algorithm 1 Brute force algorithm

```plaintext
1: procedure BRUTEFORCE
2:   iter ← 0;
3:   bestSolution ← null;
4:   Compute the Dijkstra’s algorithm between any two nodes with all weights set to 1
5:   Creates placement matrix P;
6:   SolvePlacement(P, 0);
7:   return bestSolution;
8: end procedure
```

Algorithm 2 SolvePlacement(P, m)

```plaintext
1: procedure SOLVEPLACEMENT(P, m)
2:   for n ∈ N do
3:     if n can host module m then
4:       P ← (m, n);  ▷ Respects the matrix D
5:     if m ≠ M − 1 then
6:       SolvePlacement(P, m + 1);  ▷ Places m in n
7:     else if No resources are exceeded then
8:       R ← createDependency(P);
9:       SolveRouting(P, R, null, 0, 1);
10:     end if
11:   end if
12: end for
13: end procedure
```

Algorithm 3 SolveRouting(P, R, V, x, y)

```plaintext
1: procedure SOLVEROUTING(P, R, V, x, y)
2:   r = V;  ▷ Sets r pointing to V
3:   if V = null then
4:     r = R;
5:   end if
```
Algorithm 1 Brute force algorithm (continued)
6: if \( r \) is already filled up then
7: \[ if \ V = nul \ then \]  
8: \[ V \leftarrow \text{createMigration}(P); \]  
9: \[ \text{SolveRouting}(P, R, V, 0, 1); \]  
10: \[ else \]  
11: \[ \text{iter} \leftarrow \text{iter} + 1; \]  
12: \[ \text{bestSolution} \leftarrow \text{compare}(\text{bestSolution}, \text{createSolution}(P, R, V)); \]  
13: \[ end \ if \]  
14: \[ else \]  
15: \[ if \ r[x][y−1] = r[x][N] \ then \]  
16: \[ \text{\text{\textgreater} If the previous node is equal to the destination node} \]  
17: \[ r[x][y] \leftarrow r[x][y−1]; \]  
18: \[ \text{\textgreater Sets the next node equal to the previous one} \]  
19: \[ \text{if} \ r[x] \text{is filled up} \ then \]  
20: \[ \text{SolveRouting}(P, R, V, x + 1, 1); \]  
21: \[ else \]  
22: \[ \text{SolveRouting}(P, R, V, x, y + 1); \]  
23: \[ end \ if \]  
24: \[ else \]  
25: \[ for \ n \in N \ do \]  
26: \[ if \ n \notin r[x] \text{and} \ n \text{has connection to} \ r[x][y−1] \text{and} \text{dijkstra}(n, r[x][N]) \leq N − y \ then \]  
27: \[ r[x][y] \leftarrow n; \]  
28: \[ \text{if} \ r[x] \text{is filled up} \ then \]  
29: \[ \text{SolveRouting}(P, R, V, x + 1, 1); \]  
30: \[ else \]  
31: \[ \text{SolveRouting}(P, R, V, x, y + 1); \]  
32: \[ end \ if \]  
33: \[ end \ for \]  
34: \[ end \ if \]  
35: \[ end \ procedure \]

Algorithm 2 Compare two solutions
1: procedure \text{COMPARE}(\text{best, new})
2: \[ if \ \text{best} = nul \ then \]  
3: \[ \text{return new} \]  
4: \[ else \ if \ \text{numberViolatedContraints(best)} > \text{numberViolatedContraints(new)} \ then \]  
5: \[ \text{return new} \]  
6: \[ else \ if \ \text{numberViolatedContraints(best)} < \text{numberViolatedContraints(new)} \ then \]  
7: \[ \text{return best} \]  
8: \[ else \]  
9: \[ for \ \text{objective} \ o \in \text{each defined objective} \ do \]  
10: \[ if \ \text{new.getcost(o) \times(1 + p[o]) < best.getcost(o)} \ then \]  
11: \[ \text{return new} \]  
12: \[ else \ if \ \text{new.getcost(o) \times(1 − p[o]) > best.getcost(o)} \ then \]  
13: \[ \text{return best} \]  
14: \[ end \ if \]  
15: \[ end \ for \]  
16: \[ \text{return best} \]  
17: \[ end \ if \]  
18: \[ end \ procedure \]
Algorithm 3 Random algorithm

1: procedure RANDOMALGORITHM
2:   \( \text{iter} \leftarrow 0; \)
3:   \( \text{convergenceIter} \leftarrow 0; \)
4:   \( \text{bestSolution} \leftarrow \text{null}; \)
5: Compute the Dijkstra’s algorithm between any two nodes with all weights set to 1
6: while \( \text{iter} \leq \text{MAX\_ITER} \) do
7:   \( P \leftarrow \text{GenerateRandomPlacement}(P, M); \)
8:   \( R \leftarrow \text{createDependency}(P); \)
9:   \( V \leftarrow \text{createMigration}(P); \)
10:  \( R \leftarrow \text{GenerateRandomRouting}(R, Z); \)
11:  \( V \leftarrow \text{GenerateRandomRouting}(V, M); \)
12:  \( \text{prevBestSolution} \leftarrow \text{bestSolution}; \)
13:  \( \text{bestSolution} \leftarrow \text{compare}(\text{bestSolution}, \text{createSolution}(P, R, V)); \)
14:  if \( \text{prevBestSolution} \) and \( \text{bestSolution} \) are close enough to each other then
15:     \( \text{convergenceIter} \leftarrow \text{convergenceIter} + 1; \)
16:  else
17:     \( \text{convergenceIter} \leftarrow 0; \)
18:  end if
19:  if \( \text{convergenceIter} > \text{MAX\_ITER\_CONVERGENCE} \) then
20:     return \( \text{bestSolution}; \)
21:  end if
22:  \( \text{iter} \leftarrow \text{iter} + 1; \)
23: end while
24: return \( \text{bestSolution}; \)
25: end procedure

Algorithm 4 Generate random placement

1: procedure GENERATERANDOMPLACEMENT(p, s) \( \triangleright \) Receives a matrix \( p \) and its size \( s \)
2: for \( m \in s \) do
3:   for \( n \in N \) do
4:     if \( n \) can host module \( m \) then \( \triangleright \) Respects the matrix \( D \)
5:       add \( n \) to \( \text{validNodes}; \)
6:     end if
7:   end for
8: p \( \leftarrow (m, \text{random}(\text{validNodes})); \) \( \triangleright \) Places \( m \) in a random node from \( \text{validNodes} \)
9: end for
10: return \( p \)
11: end procedure
Algorithm 5 Generate random routing

1: procedure GENERATERANDOMROUTING(r, s) \textcolor{red}{\triangleright} Receives a matrix $r$ and its size $s$
2: \hspace{1em} for $i \in s$ do
3: \hspace{2em} for $j \in [1, N - 1]$ do \textcolor{red}{\triangleright} Source, $j = 0$, and destination, $j = N$, are already filled up
4: \hspace{3em} if $r[i][j - 1] = r[i][N]$ then \textcolor{red}{\triangleright} If the previous node is equal to the destination node
5: \hspace{4em} $r[i][j] \leftarrow r[i][j - 1]$;
6: \hspace{3em} else
7: \hspace{4em} for $n \in N$ do
8: \hspace{5em} if $n$ has connection to $r[i][j - 1]$ and dijkstra($n, r[i][N]$) $\leq N - y$ then
9: \hspace{6em} add $n$ to validNodes;
10: \hspace{5em} end if
11: \hspace{4em} end for
12: \hspace{3em} $r[i][j] \leftarrow \text{random}(\text{validNodes})$;
13: \hspace{2em} end if
14: \hspace{1em} end for
15: \hspace{1em} return $r$
16: end procedure

Algorithm 6 Genetic algorithm

1: procedure GENETICALGORITHM
2: \hspace{1em} bestSolution $\leftarrow$ null;
3: \hspace{1em} generation $\leftarrow$ 0;
4: \hspace{1em} convergenceIter $\leftarrow$ 0;
5: Compute the Dijkstra's algorithm between any two nodes with all weights set to 1
6: \hspace{1em} for $i \in \text{POPULATION SIZE}$ do
7: \hspace{2em} population[$i$][$P$] $\leftarrow$ GenerateRandomPlacement(population[$i$][$P$], $M$);
8: \hspace{1em} end for
9: \hspace{1em} while generation $\leq$ MAX_ITER do
10: \hspace{2em} population $\leftarrow$ GeneticAlgorithmRouting(population);
11: \hspace{2em} prevBestSolution $\leftarrow$ bestSolution;
12: \hspace{2em} population $\leftarrow$ sort(population);
13: \hspace{2em} bestSolution $\leftarrow$ compare(bestSolution, createSolution(population[0]));
14: \hspace{2em} if prevBestSolution and bestSolution are close enough to each other then
15: \hspace{3em} convergenceIter $\leftarrow$ convergenceIter + 1;
16: \hspace{2em} else
17: \hspace{3em} convergenceIter $\leftarrow$ 0;
18: \hspace{2em} end if
19: \hspace{2em} if convergenceIter $> MAX \_ITER \_CONVERGENCE$ then
20: \hspace{3em} return bestSolution;
21: \hspace{2em} end if
22: \hspace{2em} for $i \in \text{FITTEST}$ do \textcolor{red}{\triangleright} Copies to the next generation the FITTEST individuals
23: \hspace{3em} newGeneration[$i$][$P$] $\leftarrow$ population[$i$][$P$];
24: \hspace{2em} end for
25: \hspace{2em} for $i \in [\text{FITTEST} + 1, \text{POPULATION SIZE}]$ do
26: \hspace{3em} $r1 \leftarrow$ population[random($\text{POPULATION SIZE} * p$)];
27: \hspace{3em} $r2 \leftarrow$ population[random($\text{POPULATION SIZE} * p$)];
28: \hspace{3em} newGeneration[$i$][$P$] $\leftarrow$ MatePlacement($r1[P], r2[P]$);
29: \hspace{2em} end for
30: \hspace{1em} population $\leftarrow$ newGeneration;
31: \hspace{1em} generation $\leftarrow$ generation + 1;
32: \hspace{1em} end while
33: \hspace{1em} return bestSolution;
34: end procedure
Algorithm 7 Genetic algorithm routing

1: procedure GENETIC_ALGORITHM_ROUTING(pop)
2:     for i ∈ POPULATION_SIZE do
3:         generation ← 0;
4:         convergenceIter ← 0;
5:         bestSolution ← null;
6:     for j ∈ POPULATION_SIZE do
7:         population[j][P] ← pop[i][P];
8:         population[j][R] ← createDependency(population[j][P]);
9:         population[j][V] ← createMigration(population[j][P]);
10:        population[j][R] ← GenerateRandomRouting(population[j][R], Z);
11:        population[j][V] ← GenerateRandomRouting(population[j][V], M);
12:    end for
13:    while generation ≤ MAX_ITER do
14:        prevBestSolution ← bestSolution;
15:        population ← sort(population);
16:        bestSolution ← compare(bestSolution, createSolution(population[0]));
17:        if prevBestSolution and bestSolution are close enough to each other then
18:            convergenceIter ← convergenceIter + 1;
19:        else
20:            convergenceIter ← 0;
21:        end if
22:        if convergenceIter > MAX_ITER_CONVERGENCE then
23:            break
24:        end if
25:        for j ∈ FITTEST do
26:            newGeneration[j] ← population[j];
27:        end for
28:        for j ∈ [FITTEST + 1, POPULATION_SIZE] do
29:            r1 ← population[random(POPULATION_SIZE * p)];
30:            r2 ← population[random(POPULATION_SIZE * p)];
31:            newGeneration[j][P] ← population[r1][P];
32:            newGeneration[j][R] ← MateRouting(r1[R], r2[R], Z);
33:            newGeneration[j][V] ← MateRouting(r1[V], r2[V], M);
34:        end for
35:        population ← newGeneration;
36:        generation ← generation + 1;
37:    end while
38:    pop[i] ← population[0];
39: end for
40: return pop;
41: end procedure
Algorithm 8 Mate placement

```plaintext
1: procedure MATEPlacement(p1,p2)
2:   for m ∈ M do
3:     r ← random(); ▷ Generates a random number between [0, 1]
4:       if r < α then
5:         child ← (m, p1[p]) ▷ Places m in the same node as in p1
6:       else if r < β then
7:         child ← (m, p2[p])
8:       else
9:         child[m] ← GenerateRandomPlacement(child[m], 1);
10:     end if
11:   end for
12: end procedure
```

Algorithm 9 Mate routing

```plaintext
1: procedure MATERouting(r1,r2,s) ▷ Receives two routing matrices and its size s
2:   for i ∈ s do
3:     r ← random(); ▷ Generates a random number between [0, 1]
4:       if r < α then
5:         child[i] ← r1[i]; ▷ Copies the whole path
6:       else if r < β then
7:         child[i] ← r2[i];
8:       else
9:         child[i][0] ← r1[i][0]; ▷ Copies the source node
10:        child[i][N] ← r1[i][N]; ▷ Copies the destination node
11:       child[i] ← GenerateRandomRouting(child[i], 1);
12:     end if
13:   end for
14: end procedure
```
Appendix B

Developed Graphical User Interface

Figure B.1 depicts some of the implemented features in the graphical user interface of the developed fog computing simulation tool.

Figure B.1: Developed graphical user interface.
Appendix C

Algorithm Plots

The different algorithm iteration-value plots regarding the architecture presented in Figure 4.3 for the priority set 1, priority set 2, priority set 3, and priority set 4 are presented in Figures C.1, C.2, C.3, and C.4, respectively.

(a) Genetic Algorithm.  
(b) Random algorithm.  
(c) Brute force algorithm.

Figure C.1: Iteration-value plot for the first scenario with priority set 1.
Figure C.2: Iteration-value plot for the first scenario with priority set 2.

Figure C.3: Iteration-value plot for the first scenario with priority set 3.
Figure C.4: Iteration-value plot for the first scenario with priority set 4.
Appendix D

Declaration

Figure D.1: Declaration.