Universidade Estadual de Campinas
Instituto de Computação

Judy Carolina Guevara Amaya

Quality of Service Provisioning in Fog Computing Networks

Provisão de Qualidade de Serviço em Redes de Computação em Névoa

CAMPINAS
2022

Judy Carolina Guevara Amaya

Quality of Service Provisioning in Fog Computing Networks

Provisão de Qualidade de Serviço em Redes de Computação em Névoa

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutora em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Nelson Luis Saldanha da Fonseca
Co-supervisor/Coorientador: Prof. Dr. Ricardo da Silva Torres

Este exemplar corresponde à versão final da Tese defendida por Judy Carolina Guevara Amaya e orientada pelo Prof. Dr. Nelson Luis Saldanha da Fonseca.

CAMPINAS
2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Silvania Renata de Jesus Ribeiro - CRB 8/6592

Guevara Amaya, Judy Carolina, 1984-
G939q Quality of service provisioning in fog computing networks / Judy Carolina Guevara Amaya. – Campinas, SP : [s.n.], 2022.

Orientador: Nelson Luis Saldanha da Fonseca.
Coorientador: Ricardo da Silva Torres.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Redes de computadores. 2. Computação em névoa. 3. Computação em nuvem. 4. Escalonamento de tarefas. 5. Inteligência artificial. 6. Qualidade de serviço (Redes de computadores). I. Fonseca, Nelson Luis Saldanha da, 1961-. II. Torres, Ricardo da Silva, 1977-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações Complementares

Título em outro idioma: Provisão de qualidade de serviço em redes de computação em névoa
Palavras-chave em inglês:
Computer networks
Fog computing
Cloud computing
Task scheduling
Artificial intelligence
Quality of service (Computer networks)
Área de concentração: Ciência da Computação
Titulação: Doutora em Ciência da Computação
Banca examinadora:
Nelson Luis Saldanha da Fonseca [Orientador]
Daniel Macêdo Batista
Thiago Augusto Lopes Genez
Fábio Luiz Usberti
Islene Calciolari Garcia
Data de defesa: 14-12-2022
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-1439-3584
- Currículo Lattes do autor: https://lattes.cnpq.br/4762152686250932

Universidade Estadual de Campinas
Instituto de Computação

Judy Carolina Guevara Amaya

Quality of Service Provisioning in Fog Computing Networks

Provisão de Qualidade de Serviço em Redes de Computação em Névoa

Banca Examinadora:

- Prof. Dr. Nelson Luis Saldanha da Fonseca
  IC/UNICAMP
- Prof. Dr. Daniel Macêdo Batista
  IME/USP
- Dr. Thiago Augusto Lopes Genez
  EMBL-EBI
- Prof. Dr. Fábio Luiz Usberti
  IC/UNICAMP
- Profa. Dra. Islene Calciolari Garcia
  IC/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 14 de dezembro de 2022

To José, Alberto José, Magdalena, Alberto and Diego.
The reason of my success is your unconditional love.

To all my teachers, for, as Sir Isaac Newton once said,
"If I have seen further, it is by standing on the shoulders of giants."

To all women and girls around the world who are inventing the future
and making the world a better place through the power of science.

To all people who, because of the war or the lack of opportunities,
leave their country to get an education or a job, to have better-living conditions.
Faith and perseverance are the keys to overcoming any obstacles placed in your way.

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

— William Lawrence Bragg

"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."

— Isaac Asimov

"I am among those who think that science has great beauty. A scientist in his laboratory is not only a technician, he is also a child place before natural phenomenon, which impress him like a fairy tale."

— Marie Curie

Acknowledgements

This dissertation represents the culmination of a dream. A dream that comes true thanks to the help of many people. Therefore I would like to express my deep gratitude to:

My research supervisors, Professors Nelson Fonseca and Ricardo Torres, two extraordinary human beings, admirable teachers, and passionate scientists who have become role models in both my personal and professional life. Thank you for your assistance and dedicated involvement in every step throughout my Ph.D. process, for the stimulating questions and the careful reviews of our papers. Thank you for our interesting regular face-to-face and virtual meetings, and the fascinating discussions through hundred emails about a specific topic, literally hundreds!, in short, for imparting skills that have shaped my career. Thank you for supporting me in looking for financial sources for my research. Your mentoring has meant so much to me. Thank you very much for your support and understanding over all these years.

Professors Daniel Batista, Fábio Usberti, Islene Calciolari, and Dr. Thiago Genez, my Ph.D. committee members, for their valuable suggestions and availability.

Professors of the Computer Networks Laboratory, particularly Professor Luiz Bittencourt, co-author of some of my papers, for providing insightful comments and suggestions on my investigation.

Professors Anderson Rocha, Leandro Villas, and Cecilia Rubira for their support during the pandemic situation.

Professors Flavio Miyazawa, Sandra Avila, Diego Aranha, and Lehilton Pedrosa for the conversations we had along the road about optimization, machine learning, security and computational complexity in Cloud-Fog networks.

Professors Marco Alzate, Henry Diosa, Gustavo Puerto, and Cesar Hernández, for believing in me since I was working toward my master's degree. Thank you for supporting me in the admission process to the doctoral program.

José Huamán, my beloved husband, for the love, tenderness, and patience with which he encouraged me to handle the most demanding days of this Ph.D. journey. I am so blessed to have him in my life. I am grateful for his constant dedication to our home through all the challenges of living far from our families. Thank you for being a wonderful husband and father. May God bless us so that we continue to achieve all our goals.

Magdalena Amaya and Diego Guevara, my mother and brother. Although time and distance may separate us physically, their guidance, advice, and endless love have stuck with me through it all. I would not be who I am today without them. I am sure that dad feels so proud of us.

My colleagues Luciano Chaves, Carlos Astudillo, Rodrigo Cardoso, Takeo Akabane, Hernâni Chantre, Andrei Braga, Atílio Gomes, Ray Dueñas, Natanael Ramos, Matheus Ota, Levy Gurgel, Juan Hernández, and Daniela Casas, for the memorable moments we spent together and their friendly support along the way.

Faculty members and staff of the Institute of Computing at the University of Campinas

for providing me with a favorable study environment.

Meire Ferreira, Edison Campos, Aloisio Espindola, and Gisele Camargo, for their friendship, generosity, and continued help.

The Brazilian National Council for Scientific and Technological Development (CNPq) and The World Academy of Sciences (TWAS) - for the advancement of science in developing countries, for the grant # 190172/2014-2 under the CNPq-TWAS Postgraduate Fellowship Program; Program "Formação didático-pedagógica para cursos na modalidade a distância" – UNIVESP/UNICAMP; and grant # 2021/12582-0, São Paulo Research Foundation (FAPESP), for the essential financial support to carry out this research.

Last but not least, Alberto José Guevara Huamán, our little prince, for bringing immense joy to our lives.

Resumo

Os aplicativos móveis estão em constante evolução para melhorar a experiência do usuário e, como consequência, seus requisitos de qualidade de serviço (QoS) estão cada vez mais exigentes, principalmente os relacionados ao tempo de resposta. Além disso, geram um grande volume de dados, que pode ser processado para extrair informações valiosas. No entanto, o modelo atual de processamento, a computação em nuvem, não é adequado para várias aplicações, dada a alta latência na conectividade com os dispositivos finais, o que pode levar à violação de acordos de nível de serviço (SLA) das aplicações com restrições de tempo real.

A computação em névoa endereça a limitação de provisão de baixa latência da computação em nuvem, estendendo os serviços computacionais da nuvem para mais perto dos usuários, oferecendo baixas latências, maior segurança, menor custo de largura de banda, economia de energia e maior velocidade. Esses benefícios tornam a computação em névoa uma tendência tecnológica atraente para várias aplicações, tais como realidade virtual 3D, indústria 4.0 e cidades inteligentes, incluindo telemedicina, telessaúde e transporte inteligente. No entanto, o bom funcionamento da próxima geração de aplicativos depende do desenvolvimento de mecanismos eficientes de provisionamento de QoS, os quais dão acesso à quantidade adequada de recursos exigidos para prover seus requisitos de QoS.

Esta tese de doutorado propõe mecanismos de provisionamento de QoS para computação em névoa, que combinam diferenciação de serviços com técnicas de otimização clássicas e baseadas em inteligência artificial. Como primeiro passo, analisamos os requisitos de QoS de aplicações potenciais, agrupadas em sete categorias e introduzimos um modelo de Classe de Serviço (CoS) para computação em névoa. Depois, estabelecemos uma metodologia de classificação baseada em aprendizado de máquina para identificar a classe de serviço à qual os aplicativos que chegam à rede pertencem, a fim de facilitar o escalonamento de tarefas. Propomos dois algoritmos de escalonamento de tarefas baseados em programação linear inteira (PLI) e algoritmos de aproximação para minimizar o "makespan" das aplicações. Por fim, desenvolvemos três algoritmos multiobjetivo de escalonamento de tarefas para minimizar conjuntamente o "makespan" e o custo das aplicações, considerando seus requisitos de QoS. Esses experimentos compararam o desempenho obtido dos algoritmos empregando otimização, aproximação e aprendizado por reforço. Os resultados numéricos mostraram que o algoritmo de aprendizado por reforço supera os algoritmos de programação linear inteira e de aproximação em termos da otimalidade do escalonamento e do makespan, especialmente em cenários extremos com cargas de rede variando de 60% a 95%, mantendo tempos de execução curtos.

Por meio das soluções desenvolvidas nesta tese, pretendemos viabilizar serviços de computação em névoa com QoS, reduzindo custos de gerenciamento para os administradores da rede e melhorando a interação entre os usuários e os aplicativos da Internet.

Abstract

Internet applications running on mobile devices are constantly evolving to improve user experience. Consequently, the applications' quality of service (QoS) requirements are becoming increasingly demanding, particularly those related to strict time requirements. Moreover, mobile applications generate a large amount of data that needs to be processed to extract valuable information. However, the traditional processing on Clouds is no longer adequate for real-time applications given the high latency in the connectivity between the Cloud and end devices, which may violate the Service Level Agreement (SLA) of the application.

Fog computing addresses the latency limitation of Cloud Computing by bringing Cloud services closer to the user, supporting low latency, increased security, lower bandwidth cost, energy saving, and higher speed and data privacy. These benefits make Fog Computing an attractive technological trend for several applications in the near future, such as 3D virtual reality, industry 4.0, and smart cities, including telemedicine, telehealth, and intelligent transportation. However, the proper operation of the next generation of Fog applications depends on the development of efficient QoS provisioning mechanisms that allocate the resources demanded by applications according to their QoS requirements.

This doctoral thesis proposes QoS provisioning mechanisms for Fog Computing that combine service differentiation with classical and artificial intelligence-based optimization techniques. As a first step, we analyzed the QoS requirements of potential applications and grouped them into seven well-defined categories, introducing a model of Class of Service (CoS) for Fog Computing. Then, we proposed a classifier based on Machine Learning (ML) to identify the class of service to which an application belongs to. Classifying and labeling the traffic arriving at the network edge removes the burden on the scheduler of analyzing the application's QoS requirements and provides suitable treatment for each type of service. We then proposed two task scheduling algorithms based on Integer Linear Programming (ILP) and approximation algorithms to minimize the makespan of Fog applications. Finally, we developed three multi-objective task scheduling algorithms to jointly minimize the makespan and cost of the applications considering their QoS requirements. These experiments involved applications of all classes of services and compared the performance obtained by exact and approximated optimization and a solution based on reinforcement learning. Numerical results showed that the reinforcement learning algorithm overperforms the integer linear programming and approximation algorithms in terms of the optimality of the scheduling and makespan, especially under extreme scenarios with network loads ranging from 60% to 95%, yet keeping short execution times.

The solutions developed in this thesis enable robust and flexible Fog computing, reducing management costs for network administrators while improving the interaction between users and Internet applications.

Resumen

Las aplicaciones móviles disponibles en Internet evolucionan constantemente para mejorar la experiencia del usuario y como consecuencia, sus requisitos de calidad de servicio (QoS) son cada vez más exigentes, principalmente aquellos relacionados con restricciones temporales. Además, generan un gran volumen de datos que puede ser procesado para extraer informaciones valiosas. Sin embargo, el modelo actual de procesamiento, la computación en nube, no es adecuado para varias aplicaciones, dada la alta latencia en la conectividad con los dispositivos finales, lo que podría llevar a la violación de acuerdos de nivel de servicio (SLA) de las aplicaciones con restricciones de tiempo real.

La computación en niebla trata la limitación de provisión de baja latencia de la computación en nube, extendiendo los servicios de la nube más cerca de los usuarios, ofreciendo bajas latencias, mayor seguridad, menor costo de ancho de banda, ahorro de energía y mayor velocidad. Estos beneficios hacen de la computación en niebla una tendencia tecnológica atractiva para muchas aplicaciones, tales como la realidad virtual 3D, la industria 4.0 y las ciudades inteligentes, incluyendo la telemedicina, la telesalud y el transporte inteligente. Sin embargo, el buen funcionamiento de la próxima generación de aplicaciones depende del desarrollo de mecanismos eficientes de provisión de QoS, los cuales dan acceso a la cantidad adecuada de recursos exigidos para prover sus requisitos de QoS.

Esta tesis doctoral propone mecanismos de provisión de QoS para computación en niebla, que combinan la diferenciación de servicios con técnicas de optimización clásicas y basadas en inteligencia artificial. Como primer paso, analizamos los requisitos de QoS de las aplicaciones de computación en niebla, agrupadas en siete categorías e introdujimos un modelo de Clase de Servicio (CoS) para la computación en niebla. Después, establecimos una metodología de clasificación basada en aprendizaje de máquina para identificar la clase de servicio a la que pertenecen las aplicaciones que llegan a la red, con el fin de facilitar el escalonamiento de tareas. Luego, propusimos dos algoritmos de escalonamiento de tareas basados en programación lineal entera (ILP) y algoritmos de aproximación para minimizar el "makespan" de las aplicaciones. Por último, desarrollamos tres algoritmos multiobjetivo de escalonamiento de tareas para minimizar conjuntamente el "makespan" y el costo de las aplicaciones, considerando sus requisitos de QoS. Estos experimentos comparan el desempeño de los algoritmos empleando optimización, aproximación y aprendizaje por refuerzo. Los resultados numéricos mostraron que el algoritmo de aprendizaje por refuerzo supera los algoritmos de programación lineal entera y de aproximación en términos de la optimización del escalonamiento y del "makespan", especialmente en escenarios extremos con cargas de red que van desde 60 % hasta 95 %, manteniendo tiempos de ejecución cortos.

Por medio de las soluciones desarrolladas en esta tesis, pretendemos viabilizar sevicios de computación en niebla con QoS, reduciendo los costos de gerenciamiento para los administradores de la red y mejorando la interacción entre los usuarios y las aplicaciones de Internet.

List of Figures

List of Tables

List of Algorithms

List of Acronyms

Contents

Chapter 1

Introduction

The growth in popularity of a new generation of internet applications powered by Artificial Intelligence (AI), the need for analyzing the ever-increasing volume of big data, and the evolution of Internet of Things (IoT) devices promise to intensify the rapid growth in computer networks traffic in the years to come. According to Cisco Systems [24], there will be 5.3 billion total Internet users (66% of the global population) by 2023. Besides, over 70 percent of the global population will have mobile connectivity. Of those connections, about 10% will be of the 5^{th} generation (5G) type. Moreover, 5G speeds will reach 575 Mbps, 13 times higher than the average mobile connection, offering unprecedented opportunities for mobile network operators to provide differentiated services to end-users and enterprises.

Also, globally, devices and connections are growing faster than both the population and the Internet users. By 2023, the number of devices per person will be 3.6 devices per-capita, compared with 2.4 networked devices in 2018, outnumbering humans by more than three to one. Furthermore, as shown in Figure 1.1, the IoT, through machine-to-machine (M2M) technology, will be half of the global connected devices and connections. M2M applications such as smart meters, video surveillance, healthcare monitoring, transportation, and package or asset tracking are the M2M applications that contribute most to this growth.

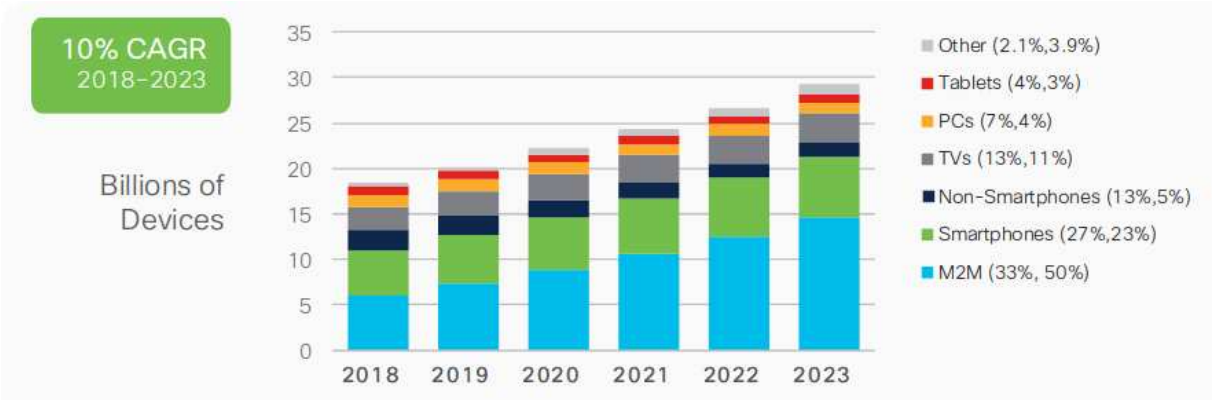


Figure 1.1: Global device and connection growth between 2018 and 2023 [24].

Furthermore, Wi-Fi will continue to gain momentum. Globally, there will be nearly

628 million public Wi-Fi hotspots by 2023, up from 169 million hotspots in 2018, a fourfold increase. Also, Wi-Fi6 hotspots will grow 13-fold from 2020 to 2023 and will be 11 percent of all public Wi-Fi hotspots.

Other important trends by 2023 are the number of downloads of mobile applications, which will reach 300 billion, with the most popular ones being social media, gaming, and business, and the increasing video definition that will be 66% of connected flat-panel TV sets.

Thus, the near future of computer networks will be marked by the growth of Internet users, devices, and connections, as well as network performance and new application requirements. This is precisely the context that has served as the motivation for this thesis.

The remainder of this introductory chapter is organized as follows. Section 1.1 describes the research topic and the related challenges that motivated the investigation in this thesis. Then, Section 1.2 states the thesis goals and the research questions proposed to be answered. After, Section 1.3 enumerates the main contributions. Next, Section 1.4 lists our publications. Finally, Section 1.5 details the thesis outline.

1.1 Motivation and problem statement

In the past decades, mobile devices have acquired more capabilities as they have evolved from 2G to higher-generation network connectivity such as 4G, LTE and now 5G. Combining device capabilities with faster, higher bandwidth and more intelligent networks have facilitated the development of advanced multimedia applications [24], which have gained enormous popularity in a wide range of areas, including gaming, e-commerce and online social network services.

Internet applications running on IoT, mobile, and end-user devices generate massive amounts of data. However, raw data by itself has very low value. Rather, the value from data processing, extracts insights and generates knowledge that can be used to enhance services.

For several years, cloud computing has been the mainstream computing paradigm. In order to process applications, tasks had to be transmitted from the edge to the cloud through the Internet's core. However, this connectivity model presents serious drawbacks such as unpredictable latency, bandwidth bottlenecks, security issues, lack of mobility support and location awareness [37].

Fog computing overcomes the inherent limitations of cloud computing, offering a geographically distributed computing architecture, with a resource pool that consists of connected heterogeneous devices, often, but not entirely located at the edge of the network, to collaboratively provide elastic computation, storage, communication, and networking services to a large scale of end users in proximity.

Although the use of fog nodes involves only brief delays in communication, several applications are still unsuitable for processing by devices located at the edge of the network like mobile devices due to various constraints, such as low processing capacity, limited memory, unpredictable network connectivity, and limited battery life [69]. Conversely,

the cloud has a large number of resources, but it use involves long communication delays. Consequently, as Chiang states in [23], it is not a binary choice between cloud and fog either, they form a mutually beneficial, inter-dependent continuum. This system, hereinafter referred to as *the cloud-fog continuum*, encompass different types of devices, allowing to support a wide spectrum of applications, from delay-sensitive to best-effort applications, but also introducing interesting scheduling challenges.

Scheduling of tasks in fog is much more complex than that of tasks on clouds due to a number of reasons. First, tasks can be moved between different physical devices such as client devices, fog nodes, and back-end cloud servers [37]. Therefore, the scheduler have to consider the diversity of capacities of the available resources along to the cloud to things (C2T) continuum [59]. Second, a fog supports a broad range of services, from mission-critical and delay-sensitive, to best-effort applications. Consequently, the scheduler needs to analyze the QoS requirements of each application before making decisions as to where tasks and virtualized resources should be run [59].

In addition, some scheduling objectives can be contradictory between them. This is the case of minimizing both makespan and the processing cost. For execution time minimization, fast hosts are preferable to slow ones, however, fast hosts are usually more expensive. Moreover, scheduling multi-task workflow on distributed platform is an NP-hard problem [31].

Three approaches can facilitate the job of the fog network scheduler: the definition of classes of service for fog computing, the classification of fog applications according to their QoS requirements, and the development of multi-objective QoS-aware task scheduling algorithms for the cloud-fog continuum. Mapping applications onto Class of Service is typical in communications network technologies that support QoS, such as LTE, 5G, and ATM networks. However, class of service definition for fog computing have not yet been defined. With regards to the classification of network traffic, numerous techniques have been proposed for the past few decades, especially for the provisioning of secure network services, but very little attention has been paid to the classification of the demands and requirements of applications and services over the Internet. Finally, although there are heuristic and meta-heuristic multi-objective algorithms with polynomial complexity capable of producing approximate or near schedules at the cost of acceptable optimality loss, they require a lot of prior experts' knowledge from specialists and human intervention, usually in terms of coding schemes [76].

Although fog computing specifies an architecture for computation, communication and storage, there is still a demand for quality of service (QoS) provisioning, especially for agile mobile services. In line with that, both industry and academia have been working on novel and efficient mechanisms to ensure QoS in fog computing. In this context, we can state the problem addressed in this thesis as follows:

> *There is a lack of QoS provisioning mechanisms for fog computing that enable providers and network administrators to offer different levels of service according to the QoS requirements of the applications. Fog computing requires QoS provisioning mechanisms to facilitate the job of task scheduler, and consequently remove the burden of the analysis of application requirements to sup-*

port the decision-making process of choosing where tasks should be processed.

QoS provisioning mechanisms provide a set of tools that can be used to manage the use of resources in a controlled and efficient manner. We are particularly interested in the integration of service differentiation, multi-objective optimization and artificial intelligence techniques to offer more robust, agile and flexible fog computing networks, reducing management costs for network administrators and enhancing users' experience.

1.2 Research aim and questions

This thesis aims to *evaluate QoS provisioning mechanisms, such as service differentiation and task scheduling, by using mathematical optimization and artificial intelligence techniques so that compliance with the QoS requirements of the applications arriving at the fog can be ensured.* To achieve this goal, we need to answer the following research questions.

Quality of Service differentiation in Fog computing: Fog computing allows the development of a variety of new and heterogeneous applications that demand a large spectrum of QoS requirements. Some of these applications are delay-sensitive [50, 66], which have associated processing deadlines, and the validity of their output depends on the time at which they become available. If a real-time application fails to meet its deadline, the consequences can range from having useless results to disastrous events [17]. Other applications supported by fog computing are non-real-time, which strive for good average-case performance and tolerate occasional slow response times. For this reason, non-real-time applications can either be processed on fog nodes or in the cloud [25]. Moreover, fog computing enables applications demanding mobility, geo-distribution, location awareness, and real-time analytics [16, 73]. In this context, we asked ourselves the following question:

- **Research question 1:** *How can fog computing provide different levels of service for selected applications according to their QoS requirements?*

Classification of fog computing applications: It is thus crucial for the efficient provisioning of services that the demands of applications arriving at the edge of the network be well understood and classified so that resources can be assigned for their processing. The mapping of applications onto classes of service (CoSs) should facilitate the matching between task requirements and resources, since labeling these tasks removes the burden of the analysis of the application requirements by the scheduler. Without a precise classification, the scheduling of application tasks and the allocation of resources can be less than optimal due to the complexity in dealing with the diversity of QoS and resource requirements. Thus, the question we must investigate is:

- **Research question 2:** *How can a fog computing network identify the class of service to which an arriving application in search of processing belongs?*

Single-objective task scheduling in the cloud-fog continuum: Fog computing is a distributed platform composed of heterogeneous networking elements, ranging from devices with limited computing capabilities, usually located at the edge of the network, to hosts with powerful processing and storage capacities placed in the cloud. Given the diversity of QoS requirements demanded by the applications in the C2T continuum, the interplay between fog and cloud plays a pivotal role in the efficient processing of applications. The major challenge in such a heterogeneous environment is the implementation of effective scheduling techniques to guarantee that applications will finish within the required time interval. In this way, we aim of answering the following question:

- **Research question 3:** *How can a scheduler decide the most suitable location for processing an application, whether on the fog or in the cloud, to meet the QoS requirements of the application and minimize the makespan?*

Multi-objective task scheduling in the cloud-fog continuum: In task scheduling, the minimization of makespan and processing cost constitute two highly desired objectives, but they are conflicting. It is also widely known that scheduling multi-task workflow on a distributed platform is an NP-hard problem [31]. Therefore, generating optimal schedules through traversal-based algorithms is extremely time-consuming. The problem becomes even more complicated when the dependencies of tasks have to be taken into account [82]. Although there are heuristic and meta-heuristic algorithms with polynomial complexity capable of producing approximate or near schedules at the cost of acceptable optimality loss, they require a lot of prior experts' knowledge and human intervention, usually in terms of coding schemes [76]. With recent advances in machine learning (ML), reinforcement learning algorithms have become increasingly versatile and powerful, facilitating their use in finding near-optimal task scheduling solutions. However, most existing contributions focus on single-objective task scheduling with service-of-level (SLA) constraints. The question therefore arises:

- **Research question 4:**

  Compared with integer linear programming and approximative algorithms, can reinforcement learning be a promising technique for task scheduling in the cloud-fog continuum to meet the QoS requirements of the application and minimize both makespan and processing cost?

1.3 Main contributions

The main contributions of this thesis are the definition of classes of service for fog computing, the establishment of a methodology for the classification of fog applications and, the development of both single and multi-objective QoS-aware task schedulers for the cloud-fog continuum. We envision that these solutions will facilitate the decision-making process for the fog scheduler through identifying the timescale and location of resources, enabling the scalable deployment of new applications. In summary, we have:

1. **Definition of classes of service for fog computing**

   Given the large spectrum of QoS requirements of fog computing applications, it is necessary to separate applications into classes to enable fog providers to offer different service levels. This issue, related to our first research question, is solved in Chapter 3 defining a set of classes of service for fog computing. Before defining classes of service, we analyze the requirements of applications on fogs. Also, in Chapter 3, we provide a mapping between the presented classes of service and the processing layers of the cloud-fog continuum. The proposed classes of service can be used to prioritize processing demands by the schedulers and resource allocation mechanisms.

2. **Establishment of a methodology for the classification of fog applications**

   Scheduling of tasks using C2T continuum resources is much more challenging than that of tasks on clouds due to the considerable heterogeneity of both demands of applications and the capacity of the devices. One possible approach to facilitate the job of the fog scheduler is to classify the applications arriving at the fog network. To address this issue, concerning to research question 2, in Chapter 4 we propose the implementation of a machine learning-based classification methodology to discriminate fog computing applications considering the QoS requirements. We also illustrate the application of this methodology in the assessment of classifiers in terms of efficiency, accuracy, and robustness to noise. The proposed methodology can be used to design effective classifiers with an output that labels the application tasks, simplifying the matching between task requirements and resources.

3. **Development of single-objective QoS-aware task scheduling algorithms for the cloud-fog continuum**

   Fog computing extends cloud services to the edge of the network. In such scenario, it is necessary to implement an efficient task scheduler to decide where applications should be executed so that their quality of service requirements can be supported. In Chapter 5, we introduce two schedulers based on Integer Linear Programming (ILP) formulation: an optimal one, CASIA-INT, and an approximate one, CASSIA-RR. The two proposed algorithms, solve our third research question scheduling tasks either in the cloud or on fog resources, while minimizing the makespan. The proposed schedulers differ from existing ones by the use of classes of service to select the processing elements on which the tasks should be executed. Moveover, the proposed scheduling algorithms also consider resources such as CPU, RAM, and storage space. Numerical results evince that the proposed schedulers outperform traditional ones, such as Random and Round Robin in terms of makespan without causing the violation of QoS requirements. Furthermore, the time required for the CASSIA-RR to generate a schedule is considerably smaller than those required by the CASSIA-INT scheduler, which is highly desirable for cloud-fog systems where fast decisions need to be made.

4. **Development of multi-objective QoS-aware task scheduling algorithms for the cloud-fog continuum**

In task scheduling, reducing the makespan and processing cost constitutes two desirable but conflicting objectives. This topic, associate to our fourth research question, is investigated in Chapter 6 through development of three multi-objective task scheduling algorithms for the cloud-fog continuum that minimize both the makespan and processing cost of the workflow, considering the QoS requirements of the applications. The FLAMSKE-INT algorithm is based on integer linear programming, FLAMSKE-RR implements an approximation to the exact solution given by FLAMSKE-INT, and FLAMSKE-RL is a reinforcement learning-based algorithm. No previous work has ever addressed the diversity of the QoS requirements of applications running on the cloud-fog continuum in a multi-objective scheduling context. We formulated the task scheduling process in cloud-fog continuum as a multi-objective optimization problem using integer linear programming (ILP), approximative algorithms, and Markov Decision Process (MDP) theory. We also compare schedules derived using classical optimization and Reinforcement Learning, and we assess the performance of the proposed schedulers using applications belonging to seven classes of service: mission-critical, real-time, interactive, conversational, streaming, CPU-Bound, and best-effort. Numerical results suggest that the FLAMSKE-RL overperforms both FLAMSKE-INT and FLAMSKE-RR in terms of optimality of scheduling and makespan, especially under network loads ranging from 60% to 95%, and yet keeping the execution times short.

1.4 Publications

In this section, we present a list of publications derived from this thesis. The list is divided into journals and conference papers.

Journals

- **Guevara, J. C.**, and Fonseca, N. L. S. da. (2021). Task scheduling in cloud-fog computing systems. Peer-to-Peer Networking and Applications, 1–16.
- **Guevara, J. C.**, Torres, R. da S., and da Fonseca, N. L. S. (2020). On the classification of fog computing applications: A machine learning perspective. Journal of Network and Computer Applications, 159, 102596.

Conferences

- **Guevara, J. C.**, Torres, R. da S., Bittencourt, L. F., and da Fonseca, N. L. S. (2022). QoS-aware Task Scheduling based on Reinforcement Learning for the Cloud-Fog Continuum. 2022 IEEE Global Communications Conference (GLOBECOM).
- **Guevara, J. C.**, Bittencourt, L. F., and da Fonseca, N. L. S. (2017). Class of service in fog computing. 2017 IEEE 9th Latin-American Conference on Communications (LATINCOM), 1–6.

1.5 Thesis outline

The remaining chapters in this thesis are organized as follows:

- Chapter 2 presents the background concepts involved in this thesis. In this chapter, we focus on the main characteristics of fog computing, reveal the differences between fog, cloud and edge computing, and describe the fog reference architecture, which was used as a basis to define the system model implemented in our experiments. We also explain the QoS provisioning mechanisms implemented in this thesis, i.e. service differentiation and task scheduling.

- Chapter 3 presents a set of classes of service for fog computing. First, we analyze the requirements of potential applications running on fogs, and then we group these requirements into classes of service. In this chapter, we also provide a mapping between the proposed classes of service and the layers of the cloud-fog continuum.

- Chapter 4 describes the adoption of a machine learning classification methodology to discriminate fog computing applications as a function of their QoS requirements. In this chapter, we also illustrate the application of this methodology in the assessment of classifiers in terms of efficiency, accuracy, and robustness to noise.

- Chapter 5 introduces two schedulers based on integer linear programming, that schedule tasks either in the cloud or on fog resources. One scheduler, called CASSIA-INT, solves an exact ILP formulation, while the other, CASSIA-RR, solves a relaxed version of the ILP formulation. Both the proposed schedulers use class of services to select the processing elements on which the tasks should be executed, and also take into consideration different types of resources in the processing devices.

- Chapter 6 proposes three multi-objective task scheduling algorithms for the cloud-fog continuum, that minimize both the makespan and processing cost of workflows, considering the QoS requirements of applications. The FLAMSKE-INT algorithm is based on integer linear programming, FLAMSKE-RR implements an approximation to the exact solution given by FLAMSKE-INT, and FLAMSKE-RL is a reinforcement learning-based algorithm.

- Chapter 7 concludes this thesis with a summary of our contributions, the directions for future work, and the list of scientific publications arising from this research process.

  As supplementary material, this thesis presents Appendix A, which reports simulation results of multi-objective QoS-aware task scheduling for all the classes of service.

Chapter 2

Fog computing networks and QoS provisioning mechanisms

The adoption of Fog-based computational resources and their integration with the Cloud introduces new challenges in resource management, which requires the implementation of new strategies to guarantee compliance with the quality of service (QoS) requirements of applications. This chapter briefly describes the fundamentals of fog computing networks and two QoS provisioning mechanisms. We first present the definition, architecture and advantages of Fog computing. We also compare Fog computing with Cloud computing and Edge computing (Section 2.1). After that, we explain service differentiation and task scheduling, two mechanisms used by the QoS provisioning mechanisms to guarantee the fulfillment of the QoS requirements of the applications (Section 2.2).

2.1 Fog computing

2.1.1 Definition, architecture and advantages

The rapid growth in adoption of Internet services is resulting in an unprecedented demand for computing resources. However, the traditional Cloud processing model is no longer feasible. Cloud has a high number of resources but involves high latency in the connectivity with end devices, which could violate the Service Level Agreement of the application if it has real-time constraints. Fog computing solves the latency limitation of the Cloud by extending Cloud services closer to the data sources.

Fog arises from the evolution of Cyber foraging [27], Mobile Edge Computing (MEC) [23], and Cloudlets [28], and can be defined as a geographically distributed computing architecture, with a resource pool that consists of connected heterogeneous devices, often, but not entirely located at the edge of the network, to collaboratively provide computing, storage, control, and networking functions closer to the users in proximity. Therefore, there are several key differences between Fog and their predecessors. First, fog computing includes devices located at Cloud, core, metro, edge, end users, and things, and not only those located at the edge of the network. Second, fog provides computing, networking and storage services along the C2T continuum, rather than considering the network edges as isolated computing platforms. Third, the fog has a horizontal architecture that supports

common computing functions for different application domains. Fourth, fog works over wireline as well as wireless networks. Consequently, end users benefit from pre-processing, geo-distribution, low latency, heterogeneity [9], and location/content awareness [13]. Additionally, fog supports different vertical domains of applications, such as those that involve complex data, real-time interactions and intensive processing. Some examples of such applications are 5G, IoT, AI, Tactile Internet, Virtual Reality (VR) and Augmented reality (AR).

In 2017, the OpenFog Consortium defined the fog reference architecture (OpenFog RA) to help unify the edge/fog ecosystem under a single, interoperable, testable set of hardware and software standards. Then, in 2018, the OpenFog RA was adopted by the Institute of Electrical and Electronics Engineers (IEEE) as the official standard for fog computing under the IEEE 1934 standard.

The fog reference architecture consist of N-tiers of nodes, as shown in Figure 2.1. Implementing a multi-tier architecture has a dual purpose: to deal efficiently with the amount of data that needs to be processed and to extract meaningful data to create more intelligence at each level. Moreover, the number of tiers varies according to the requirements of each application, including the amount and type of work required by each tier, number of sensors, capabilities of the nodes at each tier, latency between nodes and latency between sensors and actuation, and reliability/availability of nodes.

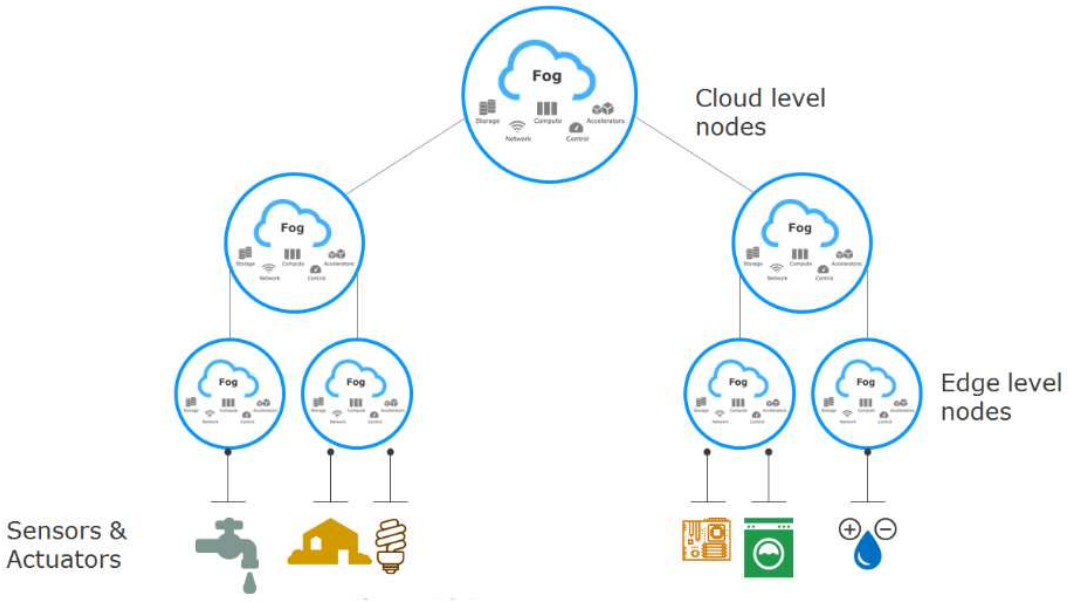


Figure 2.1: Multi-tier Deployment [61].

In the fog network architecture, the lowest-level of nodes are typically focused on sensor data acquisition, data normalization, and control of sensors and actuators. Nodes in the next higher level are focused on data filtering, compression, and transformation. They may also provide some edge analytics required for critical real time or near real time processing. Nodes at the higher levels, nearest the Cloud, are typically focused on aggregating data and turning the data into knowledge. It's important to note that the farther from the edge, the greater the insights that can be realized.

The basic structural and functional unit of a fog computing system is known as *fog node*. A fog node can be a logical or physical entity, which is be able to embed computing, storage, and networking capabilities, and is aimed at facilitating the execution of IoT applications. Among devices operating as fog nodes we can cite routers, switches, wireless access points, video surveillance cameras and servers. In addition, a fog node reusing the wireless interface can co-exist with network elements, such as a base station or a femtocell router.

Unique advantages that are potentially offered by fog can be summarized with an acronym "SCALE": Security, Cognition, Agility, Latency, Efficiency [61], explained next.

- Security – Fog brings security advantages through reducing the distance that information needs to traverse, there is less chance of eavesdropping. By leveraging proximity-based authentication challenges, identity verification can be strengthened.
- Cognition – A fog architecture, aware of customer requirements, can best determine where to carry out the computing, storage, and control functions along the C2T continuum. Fog applications being close to the end users, can be built to be aware of and closely support customer requirements.
- Agility – In order to ensure a rapid innovation and affordable scaling, fog will make it easier to create an open marketplace for individuals and small teams to use open Application Programming Interfaces (APIs), open Software Development Kits (SDKs), and the proliferation of mobile devices to innovate, develop, deploy, and operate new services.
- Latency – Real-time processing and cyber-physical system control. Fog enables data analytics at the network edge and can support time-sensitive control functions for local cyber-physical systems. This is essential for not only commercial applications but also for the Tactile Internet vision to enable embedded AI applications with millisecond reaction times.
- Efficiency – Fog can distribute computing, storage, and control functions anywhere between the Cloud and the endpoint to take full advantage of the resources available along this continuum.

2.1.2 Comparison among Fog, Cloud, and Edge computing

Cloud computing refers to the provision of on-demand and scalable computing services including processing, storage, databases, software, analytics, and intelligence, over the Internet. The main advantages that made cloud computing popular are the ability to avoid upfront investments in infrastructure, as well as the abstraction of technical details for the user. Different from Fog, that provides services locally, Cloud services are furnished from centralized large-scale data centers composed of thousands of server units. For this reason, the processing capacity in the cloud is more powerful than in the fog nodes located near to user. However, these great infrastructures require special cooling systems, and have a high energy consumption.

Today, given the increasing volume and variety of data that need to be processed, it is almost impossible to send all the data to the cloud for processing, as the network bandwidth is becoming the bottleneck of cloud computing. Moreover, response times for transfer data to cloud and back to the user are unacceptable for latency constraint applications. In contrast, the distributed architecture of Fog processes and storages data in the user's proximity, supporting location awareness and mobility, and saving bandwidth costs. As described, Cloud and Fog computing are complementary computing models, and their orchestration is necessary to meet the diversity of processing demands available on the Internet, conforming to the QoS requirements of the applications.

Edge computing is another computing model that brings cloud services to the end devices. The major objectives of edge and fog computing are similar. Both of them bring cloud computing-like capabilities to the edge of the network. They enable the computation and storage capacities within the proximity of end users to reduce service latency and save network bandwidth for delay-sensitive applications. Because of these similarities, Fog computing also is often called edge computing, but it is an erroneous interpretation, as there are key differences.

2.2 QoS provisioning mechanisms

Fog supports a diversity of new internet applications due to low latency and energy-saving. However, these applications run in a changing environment, where user mobility, among other factors, may introduce fluctuations in the availability of both bandwidth and connected hosts while processing an end device's request, which can lead to the development of misleading schedules. With the convergence of latency-sensitive applications, such as self-driving cars, control of industrial systems, and telemedicine, QoS becomes essential to deliver services in a secure, robust, and highly efficient manner. Thus, it is a must for fog computing to deploy effective QoS policies dynamically. Next, we explain two QoS provisioning mechanisms called service differentiation and task scheduling.

2.2.1 Service differentiation

Differentiation of services is an aggregate traffic handling mechanism suitable for use in large routed networks. This mechanism receives the name of "aggregated" because it aggregate or group many flows with similar QoS requirements into the same type of service, which allows to carry thousands of different applications. Given the large diversity of applications supported by fog, differentiation of services is a powerful tool to separate data flow applications arriving at network devices and to apply specific queuing treatments based on the results of the classification. This mechanism is flexible, scalable and do not require signaling, which represents significant benefits in a heterogeneous network as fog computing. Another advantage of this strategy is that the fog network administrator can define its own types of services.

The implementation of service differentiation involves two stages. First, understanding of delivery requirements of each type of application, and second, the labeling process. Both stages are studied in this thesis in Chapter 3 and Chapter 4, respectively.

2.2.2 Task scheduling

Tasks scheduling is a mechanism that determine on which resource the tasks composing an application (or request) should be processed. In fog computing, the resources may be available locally, in other fog nodes or even in the Cloud.

A scheduler produces a plan along the timeline on which tasks should be processed and defines the host where each task will be executed. To this, the schedule should consider the availability of the network resources. This includes the fluctuation of both available processing hosts and communication links. Moreover, tasks can be broken down into smaller tasks to be partially processed in the fog or the Cloud or even partially by other fog nodes; this can reduce the makespan of an application. Figure 2.2 illustrates the scheduling of an application. This figure presents a scenario in which fog node 1 receives an application composed of three tasks. Task 1 requires high computational power, which is not available in the fog layer, so it is sent to the Cloud to be processed. Task 2 also requires more resources than those available at fog node 1, but these resources are available at the fog stratum, so task 2 is forwarded to node 2 for processing. Task 3, however, requires fewer resources and can be processed locally by fog node 1. This little example illustrates how the diversity of services processed by the Cloud-Fog continuum claims for new QoS provisioning mechanisms to meet the application requirements.

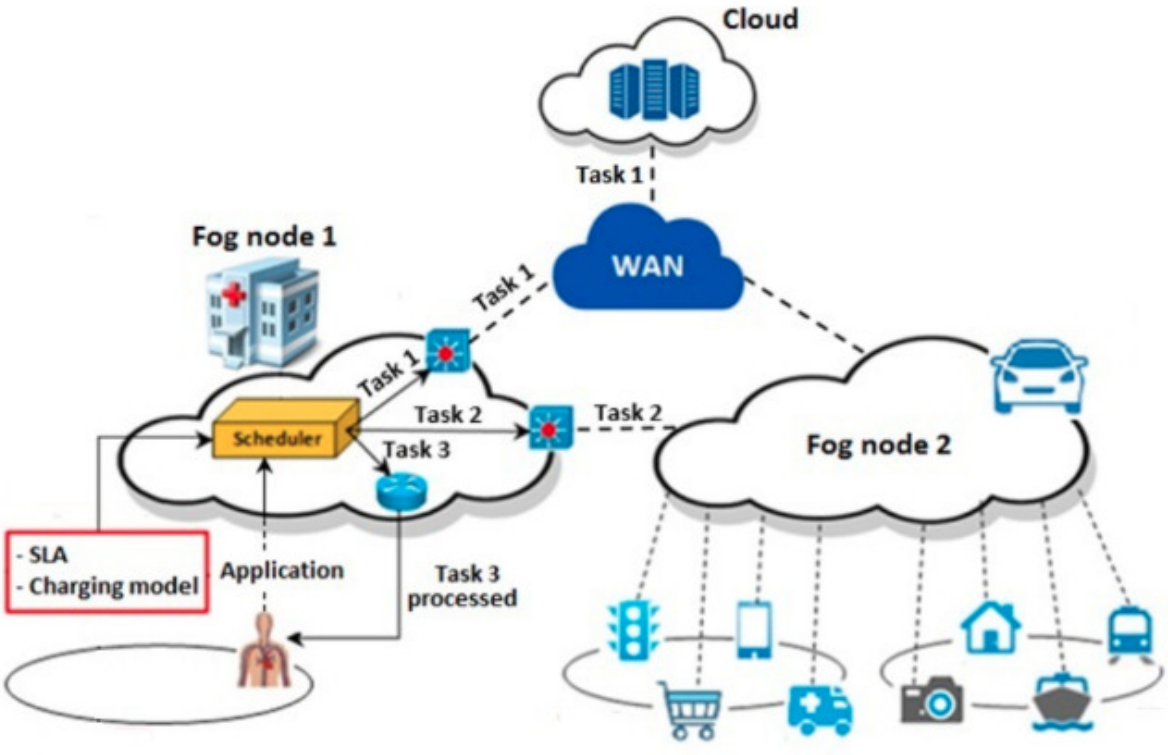


Figure 2.2: Task scheduling process in Fog computing.

Chapter 3

Quality of service differentiation for Fog computing networks

Fog computing specifies a scalable architecture for computation, communication and storage. However, there is still a demand for Quality of Service (QoS), especially for mobile services. Both industry and academia have been working on novel and efficient mechanisms for QoS provisioning in Fog computing. This chapter introduces a set of classes of service for fog computing, that takes into consideration the QoS requirements of the most relevant fog application, thus allowing the differentiation of the demands of a broad spectrum of applications. Initially, we overview related work (Section 3.1). Then, we describe the QoS requirements that best characterize Fog applications (Section 3.2). We proceed by proposing a set of classes of service for Fog computing (Section 3.3). Finally, we present a mapping between the recommended classes of service and the layers of the reference architecture presented by the OpenFog Consortium as well as some use cases (Section 3.4).

3.1 Literature review

In recent years, the research community has conducted studies that propose QoS strategies in fog computing based on service differentiation. Table 3.1 summarizes the most relevant surveys on this topic. Some of these studies were envisaged for the fog context [2, 4, 32, 34], while [71] was contemplated for the cloud-fog continuum.

Aazam et al. [4] formulate a resource management framework for fog computing based on the customer type. To build the customer profile, the model considers user characteristics such as relinquish probability of the customer, service price, the variance of the relinquish probability, and, of course, the service type. The customer profile helps determine the right amount of resources required, avoiding resource underutilization and profit-cut for the Cloud Service Provider (CSP), as well as the fog itself. The authors simulate their model by implementing multimedia and health monitoring services. Results show that more loyal Cloud Service Customers (CSCs) get better services, while for the contrary case, the provider reserves resources cautiously.

Table 3.1: Characteristics of works involving service differentiation in Fog computing networks.

Reference	Network Domain	Major contribution	Types of services implemented
Aazam et al. [4]	Fog	Customer type based resource estimation model	Video on demand, health monitoring
Souza et al. [71]	Cloud-Fog	QoS-aware service allocation based on ILP	Mice and elephant services according to the required computational capacity.
Aazam and Huh [2]	Fog	Service oriented resource management model for IoT devices, through fog.	Youtube, CloudStorage
This thesis	Fog	Definition of classes of services based on the QoS requirements of the applications	Mission-critical, real-time, interactive, conversational, streaming, CPU-bound, best-effort.

Souza et al. [71] formulate the QoS-aware service allocation problem for combined fog-cloud architectures as an integer optimization problem, to minimize the latency experienced by the services and guarantee the fulfillment of the capacity requirements. In this paper, authors use the slot as measurement unit to represent the minimum resource allocation. Moreover, services are categorized into two distinct types in terms of required computational capacity. These services are based on the so-called mice and elephants effect, where the mice represent a big amount of services with low requirements, i.e., a few number of slots, whilst the elephants represent a small number of services requiring a large number of slots. Results prove the benefits of service distribution among multiple low-delay fog nodes avoiding the high delay access on upper layers.

Aazam and Huh [2] present a service oriented resource management model for IoT devices, through fog. This study extends the model implemented in [4], by categorizing IoT devices into static, small mobile and large mobile devices providing resource management accordingly. Moreover, the model incorporates the concept of incentive for a better customer for those customers who have used more service. Results demonstrated that when characteristic of a particular customer is known, it is more justified and fair to determine and allocate resources accordingly.

Works in [2, 4, 71] evidenced the need to establish a relationship between the type of service to be processed, the QoS requirements of the applications, and the mechanisms implemented in the fog to schedule and allocate resources to the application tasks. However, the classification of fog applications and Class of service is somehow limited in these papers. Therefore, in this chapter, we propose a set of seven classes of service for fog

computing based on the QoS requirements of the applications.

3.2 Analysis of QoS requirements for Fog computing applications

Fog Computing enables new applications, especially those with strict latency constraints and those involving mobility. These new applications will have heterogeneous QoS requirements and will demand Fog management mechanisms to cope efficiently with that heterogeneity. Thus, resource management in Fog computing is quite challenging, calling for integrated mechanisms capable of dynamically adapting the allocation of resources. A very first step in resource management is to separate incoming flow of requests into Classes of Service (CoS) according to their QoS requirements.

In communications networks, network flows with similar resource demands and traffic profiles are gathered in groups that clearly identify these two. These groups, called Class of Service, are used not only to define the offering of network services, but also to identify the needs of flows for the traffic control mechanisms such as schedulers and buffer managers. The tagging of a packet by traffic classifier determines the treatment that the network flows will receive in the network core. Frameworks for Quality of Service provisioning such as IntServ and DiffServ define Class of Services as well as traffic control mechanisms to support diverse quality of service requirements of network flows.

Fog nodes will be interconnected by a Fog network composing a heterogeneous distributed system, which can federate resource with other Fogs and be integrated with other Clouds. The complexity of the Fog infrastructure and the diversity of Fog applications calls for the definition of Class of Service so that the offering of services as well as the resource demands by Fog components can be clearly identified.

Next, the quality of service requirements of Fog applications as well as other requirements will be presented by identifying the applications which will run on Fogs, especially those enabled by the deployment of Fogs.

Bandwidth – Some applications request a minimally guaranteed throughput, i.e., a Guaranteed Bit Rate (GBR). Multimedia applications are bandwidth sensitive, although some of them use adaptive coding techniques to encode digitized voice or video at a rate that matches the currently available bandwidth.

Delay sensitivity – Some applications involve a specific latency threshold, below which latency must be assured, especially for real-time applications.

Loss sensitivity – Indicates the proportion of packets which does not reach their destination.

Reliability – Is concerned with the ability of the Fog components to carry out the desired operation in the presence of many types of failure. Some applications need to have failed Fog components quickly reestablished so that tasks can be performed within some latency bounds.

Availability – Provides a measure of how often the resources of the Fog are accessible and usable upon demand by authorized entities. High availability is needed by applications and services that must be running all the time, such as mission-critical applications.

Security – Refers to the design and implementation of authentication and authorization techniques to protect personal and critical information generated by end users.

Data location – Indicates where the application data should be stored. Data can be stored locally, at the end device itself; near, or at a Fog node, or in a remote repository, in the Cloud. Requirements of data location for an application depend on factors such as response time constraints, the computational capacity of each Fog layer, and available capacity on network links.

Mobility support – Is an intrinsic characteristic of many edge devices. Continuity of the offered services should be ensured, even for highly mobile end-users. Continuous connectivity is essential for the processing needed.

Scalability – Is related to the capability of an application to operate efficiently, even in the presence of an increasing number of requests from end users. The number of users in a Fog can fluctuate due to the mobility of the users, as well as the activation of applications or sensors. Streams of data in big data processing may need to be processed within a specific time frame. The demand on Fog nodes can fluctuate and resource elasticity needs to be provided to cope with these demands.

3.3 Definition of Classes of Services for Fog computing

After identifying the requirements of most probable applications running on Fog, they were grouped in a minimum number of classes with a distinct set of requirements. Keeping the number of meaningful classes as small as possible is desirable since the complexity of allocations mechanisms is directly proportional to the number of classes. The mapping of applications into a set of classes of service is the first step in the creation of a resource management system capable of coping with the heterogeneity of Fog applications. This subsection proposes various classes of service for Fog computing: Mission-critical, Real-time, Interactive, Conversational, Streaming, CPU-bound, and Best-effort. These classes will be defined and the typical applications using these classes identified.

The first CoS to be discussed is the **Mission-critical (MC)** class. It comprises applications with a low event to action time-bound, regulatory compliance, military-grade security, privacy, and applications in which a component failure would cause a significant increase in the safety risk for people and the environment. Applications include healthcare and hospital systems, medical localization, healthcare robotics, criminal justice, drone operations, industrial control, financial transactions, ATM banking systems, and military and emergency operations.

The **Real-time (RT)** class, on the other hand, groups of applications requiring tight timing constraints in conjunction with effective data delivery. In this case, the speed of response in real-time applications is critical, since data are processed at the same time they are generated. In addition to being delay sensitive, real-time applications often require a minimum transmission rate and can tolerate a certain amount of data loss. This real-time class includes applications such as online gaming, virtual reality, and augmented reality.

The third class is denominated **Interactive (IN)**. In this case, responsiveness is crit-

ical, the time between when the user requests and actions manifested at the client being less than a few seconds. Moreover, users of interactive applications can be end devices or individuals. Examples of applications belonging to this class are interactive television, web browsing, database retrieval, server access, automatic database inquiries by tele-machines, pooling for measurement collection, and some IoT deployments.

The fourth class is the **Conversational (CO)** class. These applications include some of the video and Voice-over-IP (VoIP). They are characterized by being delay-sensitive but loss-tolerant with delays less than 150 milliseconds being perceived by humans, delays between 150 and 400 milliseconds can be acceptable, and those exceeding 400 milliseconds resulting in completely unintelligible voice conversations [51]. On the other hand, conversational multimedia applications are loss-tolerant with occasional losses causing only occasional glitches in audio or video playback, and these losses can often be partially or fully concealed [51].

The fifth class of service is **Streaming (ST)**, which releases the user to download entire files, although in potentially long delays are incurred, before playout begins. Streaming applications are accessed by users on demand and must guarantee interactivity and continuous playout to the user. For this reason, the most critical performance measure for streaming video is average throughput [51]. Additionally, streaming can refer to stored or live content. In both cases, the network must provide each flow with an average throughput that is larger than the content consumption rate. In live transmissions, the delay can also be an issue, although the timing constraints are much less stringent than those of conversational voice. Thus, delays of up to ten seconds or so from when the user chooses to view a live transmission to when playout begins can be tolerated. Examples of streaming applications are high-definition movies, video (one-way), streaming music, and live radio and television transmissions.

The sixth class is **CPU-Bound (CB)** class which is used by applications involving complex processing models, such as those in decision-making, which may demand hours, days, or even months of processing. Face recognition, animation rendering, speech processing, and distributed camera networks, are examples of CPU-Bound applications.

The final class is that of **Best-Effort (BE)**. It is dedicated to traditional best-effort applications over the Internet. For Best-effort applications, long delays are annoying but not particularly harmful; the completeness and integrity of the transferred data, however, are of paramount importance. Some examples of the Best-Effort class are e-mail downloads, chats, SMS delivery, P2P file sharing, and M2M communication.

Table 3.2 presents the relationship between the applications supported by Fog computing and the requirements of the classes of service explained above. The first column shows the recommended priority level of each class for potential adoption in scheduling systems.

Table 3.2: Classes of Service in Fog computing and their requirements.

Allocation Priority	Class of Service	Service Quality Requirements: Bandwidth	Reliability: Low	Reliability: Important	Reliability: Critical	Security: Low	Security: Medium	Security: High	Data storage: Transient	Data storage: Short duration	Data storage: Long duration	Data location: Local	Data location: Vicinity	Data location: Remote	Mobility: Low	Mobility: Medium	Mobility: High	Scalability: Low	Scalability: Medium	Scalability: High	Delay sensitivity	Loss sensitivity	Applications
1	MC	GBR			✓			✓	✓			✓			✓	✓	✓			✓	Yes	Yes	Healthcare, criminal justice, financial, biological traits, residence and geographic, military, emergency.
2	RT	GBR		✓			✓		✓			✓			✓			✓			Yes	No	Online gaming, IoT deployments, industrial control, virtual and augmented reality, interactive television, telemetry.
3	IN	GBR	✓					✓	✓				✓		✓					✓	Yes	Yes	Interactive television, object hyperlinking, web browsing, database retrieval, server access, automatic database inquiries by tele-machines, pooling for measurements collection, and some IoT deployments.
4	CO	GBR		✓			✓		✓				✓				✓			✓	Yes	No	Voice messaging, VoIP, videoconference.
5	ST	GBR		✓			✓				✓			✓		✓	✓		✓		Yes	No	Internet radio, video (one-way), high quality streaming audio.
6	CB	GBR		✓				✓		✓	✓			✓	✓				✓		Yes	Yes	Face recognition, animation rendering, speech processing, distributed camera networks.
7	BE	NGBR	✓			✓					✓			✓	✓					✓	No	Yes	Network signaling, all non-critical traffic such as TCP-based data: www, e-mail, chat, FTP, P2P file sharing, progressive video and other miscellaneous traffic.

3.4 Class of service and the Fog layered architecture

The reference architecture proposed by the OpenFog consortium in [61] provides a structural model for Fog-based computation on several tiers of nodes. As one moves further away from the edge, the overall intelligence and capacity of the system increase.

Figure 3.1 presents the system model adopted in this thesis, which consists of a distributed multi-layer architecture based on the OpenFog RA. This architecture is composed of four layers: the Cloud, at the top, a layer of end devices at the bottom, and two intermediate Fog layers. Additionally, there is a set of isolated nodes, consisting of equipment belonging to fog layers 1 and 2, intended to process mission-critical applications. The end devices send application requests to the classifier, located on the first Fog layer. The classifier identifies the CoS of the application and forwards it to the scheduler, which decides where the application should be processed, whether on the first Fog layer, on the second Fog layer, or in the Cloud.

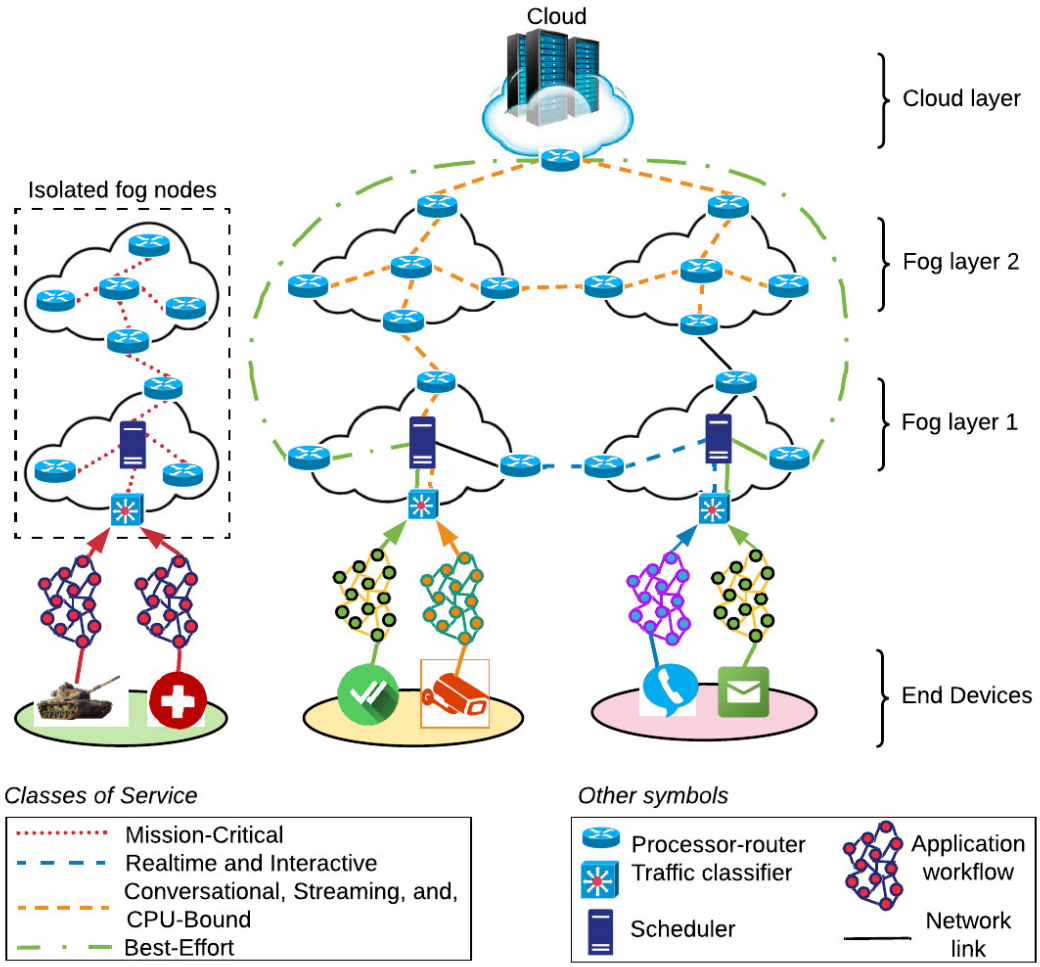


Figure 3.1: System model involving classes of service.

Table 3.3 provides a mapping between the classes of service proposed in Section 3.3, and a multi-layer Fog-Cloud architecture. Not all layers are involved in the processing of all tasks. Since Real-time, Interactive, Conversational, and Streaming applications,

such as online sensing, object hyperlinking, video conferencing, and stored streaming are delay-sensitive, these applications must be processed as close as possible to the end user, preferably at nodes located on the first and second Fog layer. CPU-bound applications require many processing resources, and for this reason, can involve all the layers of the reference architecture for the processing of tasks. Best-effort applications, such as e-mails, can be processed in the Cloud since there are no delay constraints for this class.

Table 3.3: Mapping between classes of service and processing layers of the system model.

CoS	**Service**	**Processing layer**
1	Mission-Critical	Isolated nodes
2 3	Real-time Interactive	Fog layer 1
4 5	Conversational Streaming	Fog layer 2
6	CPU-Bound	Fog layers 1 and 2, and cloud
7	Best-Effort	Cloud

Although Figure 3.1 suggests a processing layer for each CoS, a more general vision of the functionalities of each processing layer according to the class of service to be processed is provided in Table 3.4. The possibility of having a hierarchical layered system is one of the significant differences between Fog computing and edge computing. Edge computing is mainly concerned in bringing the computation facilities closer to the user; however, in a flat non-hierarchical architecture [54]. A layered architecture can introduce additional communication overhead for processing tasks at different layers. However, it has been shown that if the scheduling of tasks and resource reservations are properly carried out, processing in a hierarchical architecture can reduce communication latency and task waiting time for processing when compared to a flat architecture [22].

Table 3.4: Mapping between classes of service and network functionalities of a multi-layer Fog architecture.

Class of Service	Fog layer 1	Fog layer 2	Cloud
Mission-critical	- Support mobility. - Realize data pre-processing.	- Provide security and privacy. - Resource management. - Cost management. - Run complex jobs.	NA
Real-time Interactive	- Collect data from sensors. - Real-time data processing. - Maintain the on-board geographic information. - Provide real-time navigation.	- Realize In-depth data analysis. - Data caching. - Computation offloading. - Resource management.	NA
Conversational Streaming	- Process requests from users. - Select and cache strategic content. - Provide the most desirable services to mobile users according with their location.	- Contents caching. - Resource management. - Cost management.	NA
CPU-Bound	- Perform pre-processing. - Receive requests from users. - Monitor resource utilization. - Manage results of data query returned from nodes.	- Data processing. - Workload allocation. - Resource management. - Cost management.	- Store the inputs used for processing. - Save the results of processing. - Massive parallel data processing. - Big data management. - Big data mining. - Machine learning.
Best-Effort	NA	NA	- Process information. - Store information. - Resource management. - Cost management.

Chapter 4

Machine learning-based QoS-aware classification of fog applications

In fog computing, the scheduling of application tasks and the allocation of resources can be less than optimal due to the complexity of dealing with the diversity of QoS and resource requirements. Thus, it is crucial for the efficient provisioning of services that the demands of applications arriving at the edge of the network be well understood and classified so that resources can be assigned for their processing. In this chapter, we introduce the use of ML classification algorithms as a tool for QoS-aware resource management in Fog computing. We first overview related work (Section 4.1). Next, we introduce the implementation of a typical machine learning classification methodology with a dual purpose: to discriminate Fog computing applications as a function of their QoS requirements, and the assessment of classifiers for fog applications in terms of efficiency, accuracy, and robustness to noise (Section 4.2). Finally, we illustrate the application of the proposed methodology with a case study dealing with the classification of a dataset containing Fog application features (Section 4.3).

4.1 Literature review

Different studies have analyzed application requirements to develop service models for cloud and fog computing. In cloud computing, these studies have focused on Service Level Agreement (SLA) [7, 26, 27, 77] and Quality of Experience (QoE) management [39], while in fog computing, investigations have emphasized processing and analytics for specific applications [79], scheduling of applications to resources [21], resource estimation [3] and allocation [38, 75] for the processing of applications, and service placement [55, 70]. Next, we describe each of these proposals, which are also summarized in Table 4.1.

Alhamad et al. [7] present the nonfunctional requirements of cloud consumers. Then, based on this requirements, authors establish the most important criteria which should be considered at the stage of designing the SLA in cloud computing. Also, they investigate the negotiation strategies between cloud provider and cloud consumer and propose a method to maintain the trust and reliability between each of the parties involved in the negotiation process.

Table 4.1: Main features of works related to analysis of application requirements in Cloud and Fog computing. In the approach column, H is heuristic, E is experimental, S is simulative, and C means conceptual, i.e. that no evaluation is conducted.

Network Domain	Scope	Reference	Approach	Major contribution
Cloud	Service Level Agreement (SLA)	Alhamad et al. [7]	C	Investigate and analyze the main requirements to establish an effective model for SLA in cloud computing.
		Wu et al. [77]	H	Propose an automated cloud negotiation framework to facilitate bilateral bargaining of SLAs between a SaaS broker and multiple providers to achieve different objectives for different participants.
		Emeakaroha et al. [26]	E	Present a framework for mapping of Low-level resource Metric to High-level SLA parameters.
		Emeakaroha et al. [27]	E	Propose an application monitoring architecture that detects SLA violations at the application layer.
	Quality of Experience (QoE) management	Hobfeld et al. [39]	C	Outline the most relevant challenges of QoE management for cloud applications.
Fog	Processing and analytics for specific applications	Yang [79]	C	Present the general models and architecture of fog data streaming, by analyzing the common properties of typical applications.
	Scheduling of applications to resources	Cardellini et al. [21]	E	Evaluate a distributed scheduling algorithm that is aware of QoS attributes by using data stream processing (DSP) applications.

Continued on next page

Table 4.1: (Continued from previous page)

Network Domain	Scope	Reference	Approach	Major contribution
Fog	Resource estimation	Aazam et al. [3]	E	Devise a methodology for historical record-based resource estimation to mitigate resource underutilization and enhance QoS for multimedia IoTs.
	Resource allocation	He et al. [38]	S	Propose QoE models to evaluate the quality of service concerning both system and users.
		Wang et al. [75]	S	Study how to find the optimal placement of instances to minimize the average cost over time, leveraging the ability of predicting future cost parameters with known accuracy.
	Service placement	Mahmud et al. [55]	S	Propose a QoE-aware application placement policy for distributed computing environments, and a linearly optimized mapping of applications and fog instances that maximizes QoE.
		Skarlat et al. [70]	S	Implement the Fog Service Placement Problem (FSPP) as an ILP problem, and evaluate the placement solutions in terms of the cost of execution and QoS adherence.
	Classification of applications	This thesis	S	Provide a ML classification methodology to discriminate computing applications as a function of their QoS requirements.

Wu et al. [77] develop a Software as a Service (SaaS) broker for SLA negotiation. The aim was to achieve the required service efficiently when negotiating with multiple providers. The proposal involved the design of counter offer strategies and decision-making heuristics which considered time, market constraints and trade-off between QoS parameters. Results demonstrated that the proposed approach increases by 50% the profit and by 60% the customer satisfaction level.

Emeakaroha et al. [26] present an approach for mapping SLA requirements to resource availability, called LoM2Hi, which is capable of detecting future SLA violations based on predefined thresholds to avert these violations.

Emeakaroha et al. [27] propose CASViD, an architecture for monitoring and detection of SLA violations at the application layer, and develop tools for resource allocation, scheduling, and deployment. Authors evaluate the architecture on a real cloud testbed using three types of image rendering application workloads with heterogeneous behaviors to investigate different application provisioning scenarios and determine the effective measurement intervals to monitor the provisioning processes. Results show that the proposed architecture is efficient for monitoring and detecting single application SLA violation situations.

Yang [79] investigated common components of IoT systems such as stream analytics, event monitoring, networked control, and real-time mobile crowdsourcing, for defining an architecture for Fog data streaming.

Hobfeld et al. [39] discuss the challenges for QoE provisioning for cloud applications with emphasis on multimedia applications. This paper also presents two schemes of classification of cloud applications. The first scheme, corresponds to the traditional classification, which is guided by the service delivery model used: Infrastructure as a service (IaaS), Platform as a service (PaaS) or Software as a service (SaaS). The second scheme is a new classification proposed by the authors, which is aligned to the end-user experience and usage domain.

Cardellini et al. [21] conduct their research around Storm, an open source, scalable, and fault-tolerant data stream processing (DSP) system designed for locally distributed clusters. More specifically, in this paper, the authors evaluate a distributed version of Storm proposed by themselves. The new components added to Storm, allow to execute a distributed QoS-aware scheduler and give self-adaptation capabilities to the system. The experimental evaluation of the distributed version of Storm involves two sets of DSP applications: the former, characterized by a simple topology with different requirements, and the latter, comprising applications such as word count and log processing. Results show that the distributed QoS-aware scheduler outperforms the default centralized one, enhancing the system with runtime adaptation capabilities. However, the main limitation of this study is the instability of the scheduling algorithm that occurs when complex topologies involve many operators, negatively affecting the application availability.

Aazam et al. [3] develop a methodology, called MEdia FOg Resource Estimation (MeFoRE), to provide resource estimation on the basis of service give-up ratio, the record of resource usage and the required quality of service. The aim was to avoid resource underutilization and enhance QoS provisioning. MeFoRE methodology uses real IoT traces and traces of Amazon EC2 service.

Wang et al. [75] present an edge architecture, called mobile micro-Cloud, to provide situational awareness to processing elements. The authors introduced an approach for consistent representation of application requirements for deployment in the mobile micro-Cloud environment.

He et al. [38] investigate novel QoE-driven download joint resource allocation algorithms. First, authors formulate the QoE as a joint resource allocation problem under

different transmission rates to acquire best QoE. Then, they propose a dynamic algorithm based on shortest path tree (SPT), which is suitable for fog computing environment with content delivery frequently. The proposed models include user oriented metrics such as the Mean Opinion Score (MOS) and content popularity as well as the cost of cache allocation and transmission rate. Simulation results reveal that the benefit for using the dynamic allocation (DA) method to allocate resource can achieve high QoE performance.

Mahmud et al. [55] propose a QoE-aware application placement policy comprising of separate fuzzy logic based approaches that prioritizes application placement requests and classifies fog computational instances based on the user expectations and current status of the instances respectively. The user expectation metric includes parameters regarding service access rate of the application user, required resources to run the application and expected data processing time. By on the other hand, fog computing instances are classified according to their status metric parameters: proximity, resource availability and processing speed. Also, a linear optimization problem maps prioritized application placement requests to competent computing instances so that user QoE regarding the system services gets maximized. Experimental results indicate that the policy significantly improves data processing time, network congestion, resource affordability and service quality.

Skarlat et al. [70] evaluate the placement of IoT services on fog resources, taking into account their QoS requirements. The authors proposed an approach for the optimal sharing of resources among IoT services by employing a formal model for Fog systems. The authors introduced the Fog Service Placement Problem (FSPP) for placing IoT services on virtualized Fog resources while taking into account constraints such as execution time deadlines. Results showed that the proposed optimization model prevents QoS violations and decreases the execution cost when compared to a purely Cloud-based solution.

The related work in Fog Computing reported above concentrates on resource allocation and scheduling of applications. Most of the decisions on resource allocation and scheduling in those papers is limited to information on resource consumption by the applications. They do not consider several QoS requirements as done in this thesis. Moreover, no previous work has addressed the classification of applications on the cloud to things (C2T) continuum. Most of the work dealing with SLAs and QoS/QoE considers only the Cloud. The fog layers in C2T will increase the capacity of the system to support new applications, especially those with real-time constraints, which are not possible to be handled by the Cloud. Therefore, in [34], we introduce a methodology for classifying fog applications, which differs from the aforementioned proposals by the definition of a set of Class of Service for fog Computing and the use of machine learning algorithms to map applications onto these classes. To our knowledge, this is the first study that introduces a machine learning classification methodology to discriminate Fog Computing applications on the basis of QoS requirements.

4.2 Classification methodology based on machine learning

This section introduces a methodology for choosing and evaluating classifiers for Fog computing applications. Our classification methodology is staged in the architectural model depicted in Figure 3.1. In this scenario, users subscribe directly or indirectly to Fog infrastructure services. The first packet of a flow contains the QoS requirements of the application generating the packet flow. The proposed classifier will then map this application into a CoS using the information provided in the first packet. Alternatively, the first packet could already carry the CoS of the application. However, such an option would make rigid the CoS adopted by the Fog provider, preventing the redefinition of this CoS for the handling of new applications with unique QoS requirements.

Classification techniques based on ML aim at mapping a set of new input data to a set of discrete or continuous valued output. Fig. 4.1 summarizes the key steps in the building of a classifier of Fog applications based on ML algorithms. In this thesis, the classification steps were executed offline. Indeed, the best performing classifier evaluated in these steps can be executed on-line in an operational Fog.

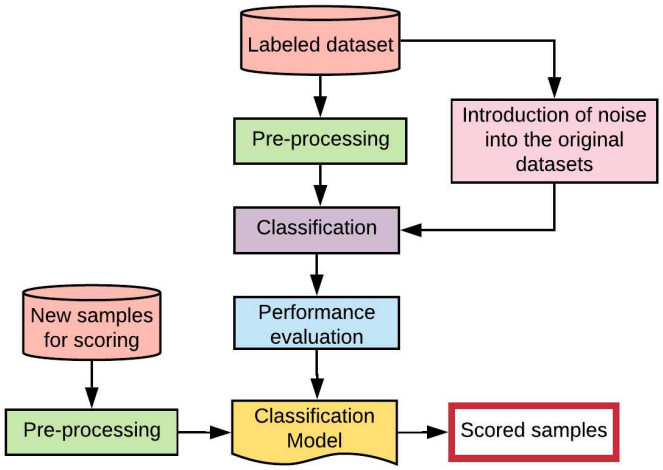


Figure 4.1: ML-based methodology for the classification of Fog computing networks.

The first step is the creation of a *labeled dataset* containing QoS attributes of Fog applications, which can be either real or synthetic. A real dataset is one collected from a system in operation while a synthetic one involves data collection generated by models. Real-world datasets usually contain sensitive data [57] and are often unavailable for the maintenance of the user information. Thus, the use of synthetic data sets is quite common, especially in the studies of systems yet to be built.

Since the value of QoS attributes differs widely, these values should be *pre-processed* to produce compatible ranges of values for classification. Pre-processing includes operations for data transformation, which standardize and consolidate data into more appropriate forms for classification, while data reduction includes the selection and extraction of both features and examples in a database [30, 72]. Data normalization avoids the handling of an attribute which has large values that dominate the results of the classification, thus improving the predictive power of the model. Feature selection, on the other hand, removes

redundant and irrelevant data from the input of the classifier, without compromising critical attribute information [72].

Noise is an unavoidable problem in collecting data from real-world systems. It can change the knowledge extracted from the data set and affects the accuracy of the classification, building time, size, and interpretability of the classifier [83, 84]. Common sources of noise are channel capacity fluctuation, fluctuation in the availability of computational resources, imprecision inherited from measurement tools, and the inability to accurately estimate the true demands of applications.

In such noisy scenarios, robustness is considered more important than performance because robustness allows a priori knowledge of the behavior expected from a learning method despite noise when it is unknown [30]. Robustness [42] is defined as the capability of an algorithm to be insensitive to data corruption and, consequently, more resilient against the impact of noise. A copy of the original dataset should be contaminated by the *introduction of noise* at different levels to check the robustness of a classifier. In this thesis, uniform attribute noise levels of 10%, 30%, and 50% are employed. The performance of the classifiers which learned from the original data set is compared to that of those which learned from a noisy data set. The most robust classifiers are those which learned from noisy data sets yet produced results similar to those learned from a noise-free data set [30].

Classification techniques based on ML can then be applied. The performance of the classifier should be assessed by a *performance evaluation process*, which encompasses both the measurement of performance and the result of statistical tests. The adequacy of performance is usually assessed by metrics such as accuracy, efficiency, and robustness. Statistical testing gathers evidence of the extent to which an evaluation metric on the resampled data sets is representative of the general behavior of the classifier [60].

At this point, the *classification model* is ready to receive new input for scoring. The new data, however, must also be subjected to a *pre-processing* process.

4.3 Classification of Fog computing applications: A case study

This section illustrates step by step the methodology introduced in the previous section for the classification of Fog applications using the CoS presented in Chapter 3. Moreover, an example of a Decision Tree that classifies Fog computing applications from the values of their QoS requirements is provided at the end of this section.

4.3.1 Labeled dataset

To train and test the classifiers employed in this thesis, we built a dataset[1] composed of 14,000 mutually exclusive applications generated from data in the intervals of values acceptable for each QoS requirement of the application. 90% of the data were reserved for training, while the remaining 10% were used for testing. It was assumed that each

[1]publicly available at http://bit.ly/34x6X1O

incoming application had additional fields containing nine QoS requirements, from now on referred to as "attributes": Bandwidth, Reliability, Security, Data Storage, Data location, Mobility, Scalability, Delay sensitivity, and Loss sensitivity.

Table 4.2 shows the range of QoS requirement values for each class of service: Bandwidth [39, 51], Reliability [19], Security [47], Data storage [7, 39], Data location [7, 19, 39], Mobility [19], Scalability [7, 19, 39], Delay sensitivity [18, 39], Loss sensitivity [9]. These ranges were used to generate the synthetic dataset of Fog applications and were employed for training and testing samples to evaluate the classifiers in this thesis.

Table 4.2: Intervals of the QoS requirements for Fog computing.

QoS Requirements	**Nominal Categories**	**Intervals**	**Class of Service**						
			MC	**RT**	**IN**	**CO**	**ST**	**CB**	**BE**
Bandwidth (Mbps)	Low	$0 < x \leqslant 1$	✓		✓	✓			✓
	Medium	$1 < x \leqslant 5$					✓	✓	
	High	$5 < x \leqslant 1000$		✓					
Reliability	Low	$x = 1$			✓				✓
	Important	$x = 2$		✓		✓	✓	✓	
	Critical	$x = 3$	✓						
Security	Low	$x = 1$							✓
	Medium	$x = 2$		✓		✓	✓		
	High	$x = 3$	✓		✓			✓	
Data storage (h)	Transient	$0 < x \leqslant 1$	✓	✓	✓	✓			
	Short duration	$1 < x \leqslant 730$						✓	
	Long duration	$730 < x \leqslant 8760$					✓	✓	✓
Data location (ms)	Local	$0 < x \leqslant 10$	✓	✓					
	Vicinity	$10 < x \leqslant 20$			✓	✓			
	Remote	$20 < x \leqslant 100$					✓	✓	✓
Mobility (Km/h)	Low	$0 < x \leqslant 5$	✓	✓	✓			✓	✓
	Medium	$5 < x \leqslant 25$	✓				✓		
	High	$25 < x \leqslant 100$	✓			✓	✓		
Scalability (No. of IoT users/ end users)	Low	$0 < x \leqslant 60$		✓					
	Medium	$60 < x \leqslant 120$					✓	✓	
	High	$120 < x \leqslant 200$	✓		✓	✓			✓
Delay sensitivity (Interaction latency in ms)	Low	$1000 < x \leqslant 100000$					✓	✓	✓
	Moderate	$10 < x \leqslant 1000$			✓	✓			
	High	$0 < x \leqslant 10$	✓	✓					
Loss sensitivity (PELR)	Low	$10^{-3} < x \leqslant 10^{-2}$				✓			
	Moderate	$10^{-6} < x \leqslant 10^{-3}$		✓			✓		
	High	$0 < x \leqslant 10^{-6}$	✓		✓			✓	✓

Attribute values were assigned by employing a uniform probability distribution, within the intervals specified for each CoS in Table 4.2. An independent random number generator randomly created the values of each attribute. Transient data were removed according to the Moving Average of Independent Replications procedure [45]. Attribute values were made up of safe and borderline examples. Safe examples were placed in relatively homogeneous areas concerning the class label. Borderline examples, on the other hand, are located in the area surrounding class boundaries, where different classes overlap. Also, to estimate the robustness of the classifiers, the third group of attribute values, called noisy examples, was generated. The term noisy sample will be used in this thesis to refer to the samples generated to represent the corruption of their attribute values.

Fig. 4.2 illustrates the safe samples, labeled as S, the borderline examples, labeled as B, and the noisy samples, labeled as N. The continuous line shows the decision boundary between the two classes.

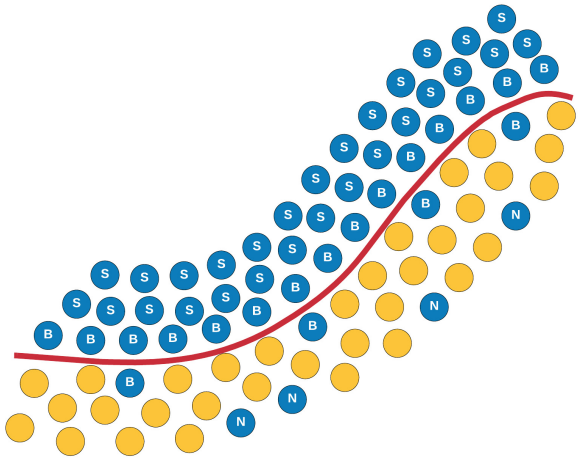


Figure 4.2: The three types of samples considered in the generation of the labeled dataset: safe (S), borderline (B), and noisy (N).

4.3.2 Pre-processing

Z-score normalization was used to adjust attribute values defined on a different scale. Mean and standard deviation were computed on the training set, and then, the same mean and standard deviation were then used to normalize the testing set.

We reduced the number of input attributes to be used by classification algorithms. This process, known as dimensionality reduction, removes irrelevant, redundant, and noisy information from the data, often leading to enhanced performance in learning and classification tasks [67]. Two techniques can be used for dimensionality reduction: one, by using feature selection techniques such as Relief-F [52], CFS [36], MCFS [20], and the Student's t-test, which rank the given feature set so that the least significant features can be removed from the problem. The second way involves feature extraction techniques, such as Principal Components Analysis (PCA), which creates new features from the given feature set. The resulting number of the features is less than that the initial set of features.

In this thesis, the significance level of the impact of each input attribute on the system is determined utilizing a PCA, guided further by a correlation analysis and the semantics of the Fog computing environment. The correlation analysis and complementarity with the PCA are explained below.

The correlation analysis is a statistical evaluation technique used to study the strength of a dependence between pairs of attributes. Fig. 4.3 provides a graphic representation of the correlation matrix among dataset attributes for Fog applications.

The correlation matrix shows that there is a statistical association of more than 50% between the following variables: "Data storage" and "Data location" (0.623), "Data storage" and "Delay sensitivity" (0.596), "Loss sensitivity" and "Mobility" (0.563), "Bandwidth" and "Scalability" (-0,605) and, "Data location" and "Delay sensitivity" (0.674). The symbol "-" in the correlation value between the attributes of "Bandwidth" and "Scalability" indicates an inverse relationship between the two.

Principal components analysis (PCA) [63] is a common approach for dimensionality reduction that uses techniques from linear algebra to find new attributes, denominated

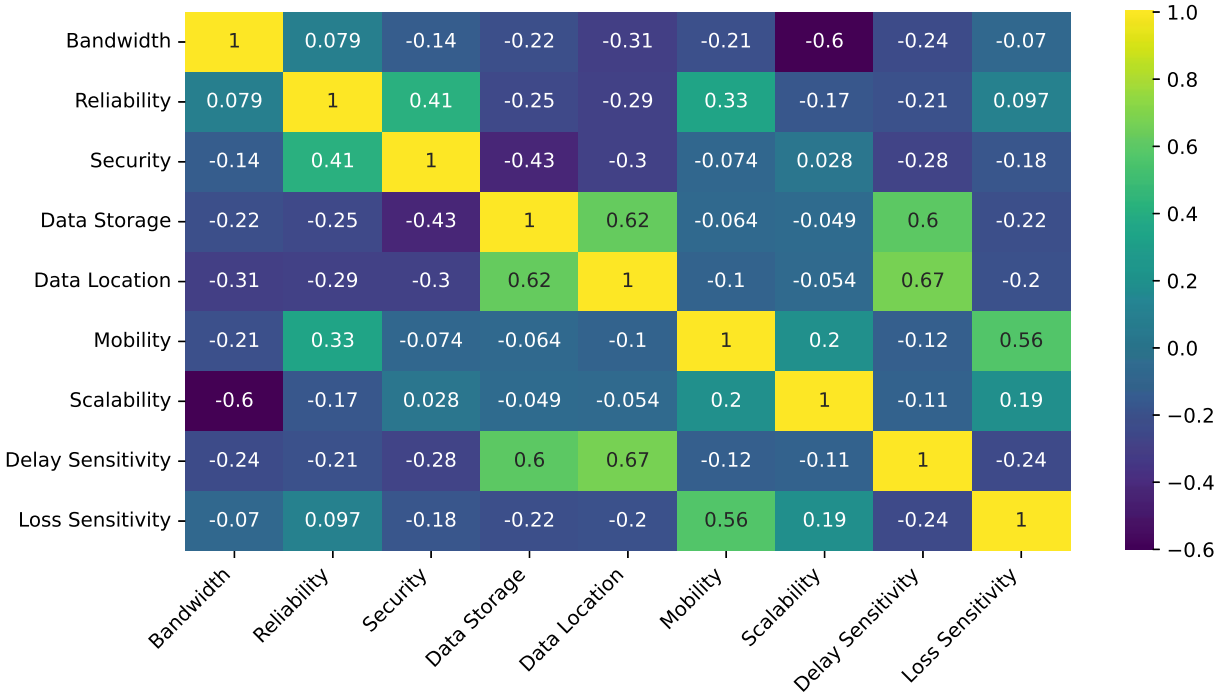


Figure 4.3: Attribute association estimates.

principal components, which are linear combinations of the original attributes. They are orthogonal to each other, and capture the maximum amount of variation in the data. Fig. 4.4 shows the scree plot of the percent variability explained by each principal component.

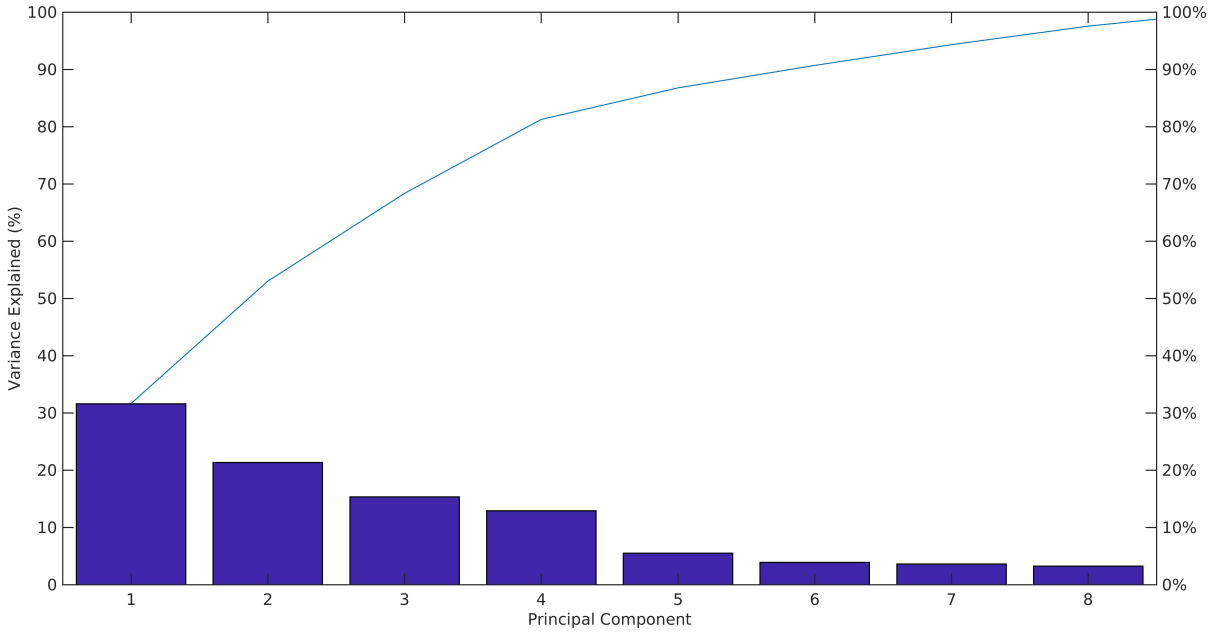


Figure 4.4: Percent variability explained by each principal component.

As illustrated by Fig. 4.4, the first seven principal components explain 94.314% of the total variance. The first component by itself explained less than 35% of the variance, so more components might be needed. Also, Fig. 4.4 reveals that the first three principal components explain roughly two-thirds of the total variability in the standardized ratings.

In addition to the percent variability explained by each principal component, all nine attributes were represented in a bi-plot by a vector. The direction and length of the vector indicate the contribution of each attribute to the two principal components in the plot. For instance, Fig. 4.5 shows the coefficients of each attribute concerning the first two principal components.

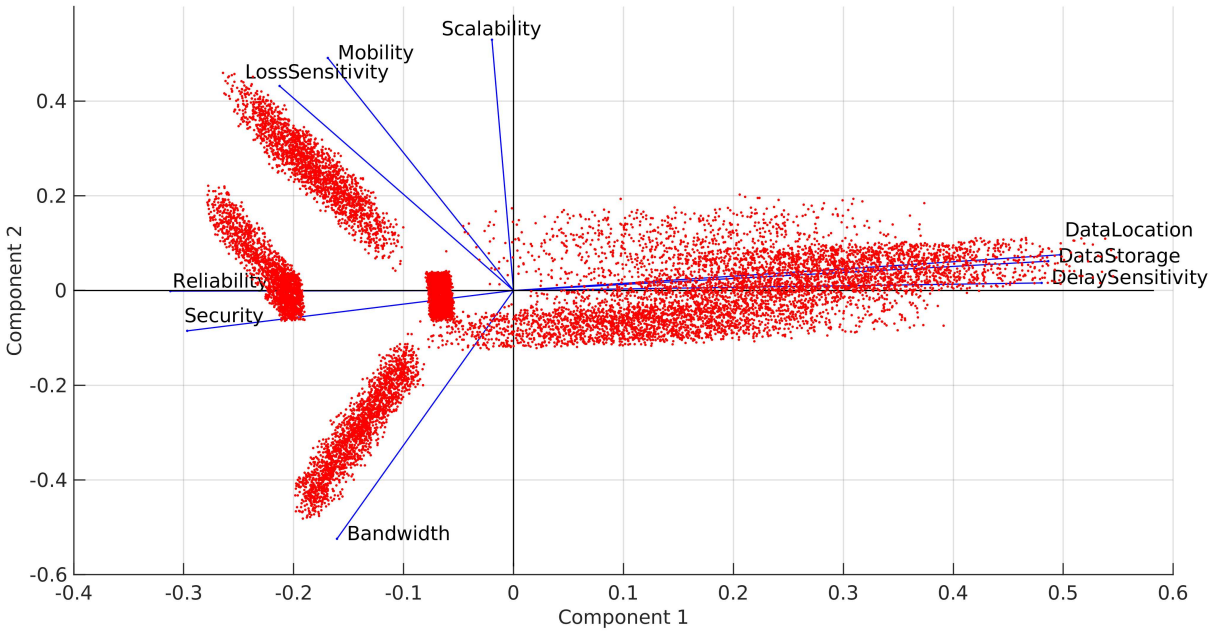


Figure 4.5: Orthonormal principal component coefficients for each variable and principal component scored for each observation (principal components 1 and 2).

The other five principal components were also plotted in bi-plots. Table 4.3 shows the contribution of the attributes to each principal component.

Table 4.3: Attribute coefficients for each principal component.

Attribute	**Principal Component**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Bandwidth	-0.161	**-0.524**	-0.449	-0.065	0.079	0.118	0.114
Reliability	-0.312	-0.001	0.040	**0.702**	0.313	0.295	-0.318
Security	-0.297	-0.085	**0.595**	0.283	-0.342	-0.286	-0.021
Data Storage	**0.486**	0.062	-0.099	0.181	**0.490**	-0.375	-0.389
Data location	**0.497**	0.075	0.022	0.213	-0.351	-0.283	0.062
Mobility	-0.169	**0.491**	-0.327	0.350	0.117	-0.207	**0.645**
Scalability	-0.020	**0.530**	0.337	-0.366	0.309	0.318	-0.014
Delay sensitivity	**0.480**	0.016	0.016	0.295	-0.273	**0.673**	0.134
Loss sensitivity	-0.213	0.432	-0.460	-0.044	**-0.481**	0.011	**-0.544**

Interpretation of the principal components is based on finding which variables are most strongly correlated with each component, that is, which of these numbers is large, the farthest from zero in either direction. The decision as to what values should be considered large is a subjective one and reflects knowledge of the system under evaluation. It was determined that a correlation value was relevant to our study when it was above 0.48, since this value is the largest within each principal component, and most of the variables

having this high value are highly correlated, as shown by the components of the correlation matrix. These large correlation values are in boldface in Table 4.3.

The principal component results can be interpreted with respect to the value deemed to be significant. The first principal component correlated strongly with three of the original attributes. Thus, the first principal component increases with increasing Data storage, Data location, and Delay sensitivity, suggesting that these three attributes vary together.

On the other hand, the coefficients belonging to the second principal component show that the behavior of the feature bandwidth opposes that of the behavior of the features Mobility and Scalability. This reflects the fact that greater mobility is associated with changes in the network topology, which, in turn, increases the fluctuations in communication links and reduces the bandwidth availability. Moreover, since bandwidth is a finite resource, if the number of users connected to the Fog increases the rate at which each user transmits and receives data decreases.

Other features that reveal an inverse relationship are Data Storage and Loss sensitivity, in the fifth principal component, and mobility and loss sensitivity in the seventh principal component.

Finally, the third, fourth, and sixth principal components increase with only one of the values, that is, there is only one variable with a value 0.48 or higher. These variables are Security, Reliability, and Delay sensitivity, respectively. Accordingly, the third, fourth, and sixth principal components can be interpreted as measures of how necessary the use of isolated nodes is to process the application, how quickly failed Fog components should be reestablished, and how sensitive the Fog application is to the delay.

Based on the analysis described above and considering the semantics of the case study about which variables deserve more attention into the Fog computing environment, redundant attributes such as Data location, Data storage, and Mobility have been removed. Thus, six of the original nine attributes were selected and maintained for the stage of classification. These were Delay sensitivity, Scalability, Loss sensitivity, Security, Reliability, and Bandwidth.

4.3.3 Classification

Seven classifiers are evaluated for potential adoption: Adaptive neuro-fuzzy inference system from data, using subtractive clustering (ANFIS), Decision Tree (DT) Artificial neural network with 2 hidden layers, trained with the Levenberg-Marquardt backpropagation algorithm, (ANN(1)); Artificial Neural Network with 1 hidden layer, trained with the algorithm Scaled conjugate gradient backpropagation (ANN(2)); Artificial Neural Network with 2 hidden layers, trained with the algorithm Scaled conjugate gradient backpropagation (ANN(3)); K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). These algorithms have different characteristics with respect to noise sensitivity, the speed of learning, and the speed of prediction. For instance, SVMs are known to be very accurate but also sensitive to noise [30]. ANNs predict rapidly, but the speed of learning is low, while KNN provides rapid learning, but are considered difficult to interpret [56]. These classifiers are typically employed in the context of complex recognition or prediction ap-

plications, especially when there is a lack of labeled samples, a lack of time for training and testing, or even a lack of appropriate hardware for timely assessment of the quality of classification models. These same conditions are present in typical Fog computing scenarios, as shown in Fig. 3.1, in which heterogeneous devices with limited computing and storage capabilities must interoperate in real time to ensure compliance with the QoS requirements of time-sensitive applications. The investigation of data-driven approaches (e.g., based on deep learning models) for constrained processing scenarios [41, 43, 68] as in our target application is left for future work.

4.3.4 Performance evaluation

This section focuses on two main subtasks of the evaluation process: measurements of the performance and statistical significance of the performance metrics. The hardware configuration used for both the dataset generation and the experiments carried out was: Intel Core i7 processor, 12GB of RAM, 1 TB of storage and Windows 10 operating system. First, the performance of each classifier is evaluated under ideal conditions, that is, without noise. Each classifier is assessed by measuring its accuracy and efficiency. Then, the performance of each classifier is assessed in the presence of noise. In this case, the robustness of each classifier in two different scenarios is assessed: when noise is introduced into the training set, and when noise is introduced in the testing set. In both cases, with and without noise, Wilcoxon's signed-rank test is employed to determine whether or not the difference in accuracy between two classifiers is statistically significant. Inserting noise in the training data set is designed to evaluate the robustness of the classifier trained by the specific data set. Inserting noise in the test data set aims at assessing the robustness of a classifier in the face of data sets with no evident data relationship, for which data relationships are not so evident.

A stratified *10*-Fold Cross-validation (*10*-FCV) [49] protocol was adopted. In this protocol, there are 10 partitions, each with the same proportion of samples belonging to each class. Nine of them are used for training, and the remaining fold is used for testing. This process is repeated 10 times, with each of the 10 folds used exactly once as the testing data. Finally, the 10 results obtained from each one of the test partitions are combined to produce a single average value representing accuracy. Table 4.4 summarizes the results obtained from the accuracy estimation for each classifier assessed without noise.

Table 4.4: Accuracy estimation results.

Classification Technique	**Testing**	
	Average Accuracy (%)	**Average Time (s)**
ANFIS	99.271	0.048
DT	99.986	0.029
ANN(1)	100	0.032
ANN(2)	100	0.035
ANN(3)	100	0.031
KNN	100	0.073
SVM	100	0.044

Table 4.4 shows that the average accuracy results for the testing process were close to 100%, except for the ANFIS was only 99.2% accurate. The shortest prediction time was obtained with the DT algorithm, while the KNN took the longest time. One possible explanation is that the DT algorithm divides the input space, matching the way the attributes were defined and assigned to each CoS by using intervals. The way the synthetic dataset was created may have led to application instances with properties which were easy to model, thus boosting the performance of the evaluated classifiers.

A two-tailed Wilcoxon signed-rank test with a significance level of 0.05 was also applied to the observations obtained from the different classifiers with no attribution of noise. The results revealed that the accuracy level of the classification obtained from the DT, ANN(1), ANN(2), ANN(3), KNN, and SVM were approximately the same. In contrast, observations obtained from the ANFIS reveled statistically significant differences in relation to the observations obtained from the other classifiers.

An attribute noise was introduced into the original dataset to check the effect of noise on the classifiers. Corrupting the data impacts directly on the significance of the attributes for the purpose of classification [30]. Moreover, attribute noise includes erroneous attribute values, which is one of the most common types of noise in real-world data [84].

Noise is introduced into each partition in a controlled manner, i.e., a certain percentage of the values of each attribute in the dataset were altered. To this end, the steps employed by the authors in [30] were followed. To corrupt each attribute A_i, a certain percentage of the examples in the dataset were chosen, and the A_i value of each was assigned a random uniform value from the domain D_i of the attribute A_i.

Noise was introduced into the training partitions to create a noisy data set from the original, as follows:

(i) Noise as a percentage of the original value was introduced into a copy of the full original dataset.

(ii) The two datasets, the original and the noisy copy, were partitioned into 10 equal folds, i.e., each with the same number of examples of each class.

(iii) The training partitions are built from the noisy copy, whereas the test partitions were formed from examples from the noise-free dataset.

Noise was introduced into the testing partitions following the same steps described above, except that in Step *iii* the testing partitions are built from the noisy copy, while the training partitions were formed from examples from the noise-free data set.

The methodology followed in this subsection presents two differences concerning the methodology described in [30]. First, the one proposed here includes the same number of examples of each class in the percentage of values of each attribute in the data set to be corrupted. Second, the introduction of attribute noise was extended to a second simulation scenario in which accuracy and robustness were estimated for each one of the individual classifiers using a clean training set and a testing set with attribute noise. Introducing attribute noise into the training set while maintaining the testing set clean enabled the

assessment of the capacity of the classifier to deal with problems in the training stage, such as overfitting. Overfitting occurs when the model is too tightly adjusted to data offering high precision in known cases, but behaving poorly with unseen data. Conversely, introducing attribute noise into the testing set while maintaining the training set clean, enables us to assess the robustness of the trained model.

After introducing noise, the accuracy of classifiers is determined by means of 5 runs of a stratified 10-Fold Cross-Validation (FCV). Hence, a total of 50 runs per dataset, noise type, and level are averaged. Ten partitions make the noise effects more notable, since each partition has a large number of examples (1,400). The robustness of each algorithm is then estimated by using the Relative Loss of Accuracy (RLA) given by Equation 4.1 is:

$$RLA_{x\%} = \frac{Acc_{0\%} - Acc_{x\%}}{Acc_{0\%}}, \tag{4.1}$$

where $RLA_{x\%}$ is the relative loss of accuracy at a noise level of $x\%$. $Acc_{0\%}$ is the test accuracy in the original case, that is, with 0% of induced noise, and $Acc_{x\%}$ is the test accuracy with a noise level $x\%$.

Next, the robustness of the classifiers when the noise has been introduced for the two mentioned scenarios will be evaluated.

Classification using a training set with attribute noise and a clean testing set

Table 4.5 shows the average performance and robustness results for each classification algorithm at each noise level, from 0% to 50%, on training datasets with uniform attribute noise.

Table 4.5: Test accuracy and RLA results of classifiers trained with noisy datasets.

	Noise Level (%)	ANFIS	DT	ANN(1)	ANN(2)	ANN(3)	KNN	SVM
Test accuracy results	0	99.271	99.986	100.000	100.000	100.000	100.000	100.000
	10	94.429	99.986	99.911	99.950	99.921	99.800	99.997
	30	84.900	99.950	99.676	99.800	99.691	99.029	99.836
	50	78.564	99.979	90.807	91.370	91.419	99.193	99.600
RLA values results	0	-	-	-	-	-	-	-
	10	0.04878	0.00000	0.00089	0.00050	0.00079	0.00200	0.00003
	30	0.14477	0.00036	0.00324	0.00200	0.00309	0.00971	0.00164
	50	0.20859	0.00007	0.09193	0.08630	0.08581	0.00807	0.00400

As can be observed in Table 4.5, the DT is the most robust classifier for all noise levels. On the other hand, the ANN(1), ANN(2), and ANN(3) present high robustness for noise levels (10-30%). Conversely, the RLA of classifiers based on neural networks rises linearly to 9% when the noise level is 50%. The least robust classifier is the ANFIS, for which the loss of accuracy increases exponentially as the proportion of noise level rises, to the point that when the noise level is 50%, its RLA is above 21% of that a clean dataset.

Fig. 4.6 shows the accuracy ratio and testing time when training takes place with both clean datasets, and those disrupted by uniform attribute noise levels of 10%, 30%, and 50%. A marker identifies each classification algorithm, and a different color identifies each noise level. The light- bands indicate the areas of the greatest accuracy or the slowest

testing times, and the light-purple intersection of these bands indicates the area where the best results for both accuracy and testing time are found.

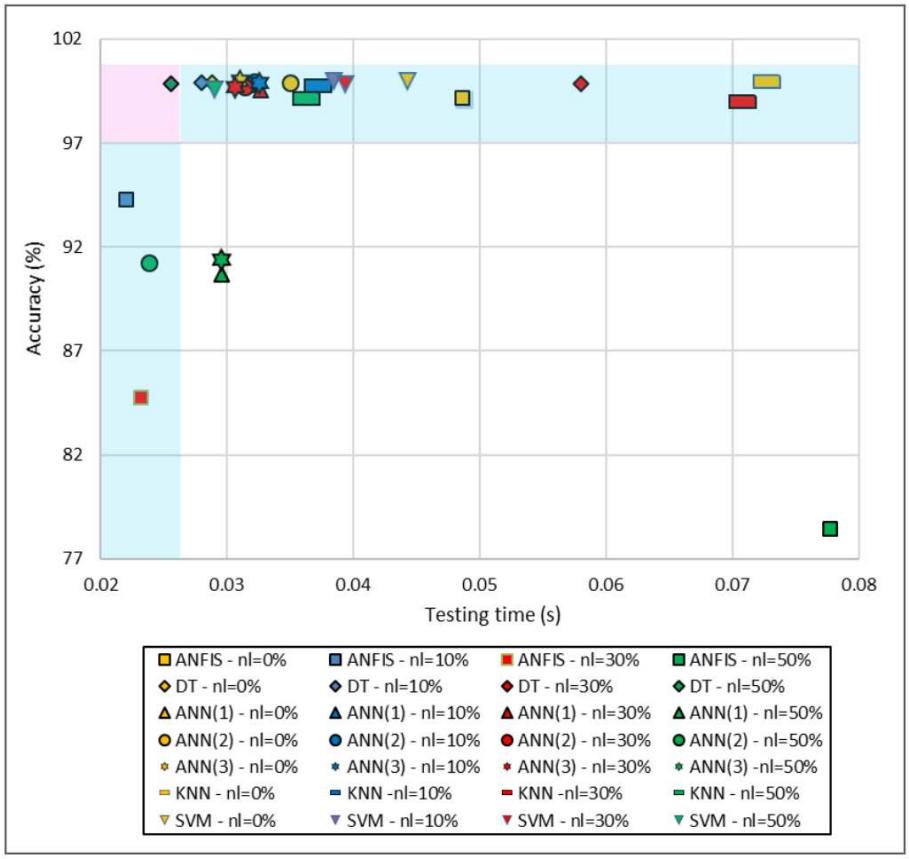


Figure 4.6: Accuracy rates concerning the testing time for classifiers trained with both clean and noisy datasets.

The DT algorithm takes only 25 milliseconds for classification with the greatest accuracy for up to 1,400 applications simultaneously arriving at the edge, when training has taken place using datasets with a uniform attribute noise level of 50%.

Table 4.6 presents the results of a two-tailed Wilcoxon signed-rank test (considering a significance level of 0.05) to verify statistical differences between the accuracy results of classifiers trained with noisy datasets. Each cell shows the results of the statistical tests between a single classifier with the others for the four levels of noise *(nl)*. Left ' $\leftarrow$' and up '$\uparrow$' arrows indicate the most accurate, while an empty cell refers to "no statistical difference between the pairs of classifiers" in that row and column.

Table 4.6 shows the effect of noise data sets in training. The performance of ANFIS is the least accurate. ANN(1), ANN(2), and SVM produce a similar performance while the DT outperform most of the results of the rest of classifiers when it is trained using datasets with an attribute noise level equal to or greater than 30%.

Classification using a clean training set and a testing set with attribute noise

Table 4.7 shows the results for average performance and robustness for each classification algorithm at each noise level, from 0% to 50%, from the testing of datasets with uniform

Table 4.6: Statistical test for the accuracy of classifiers trained with noisy datasets. nl denotes the percentage of noise level present in the training dataset.

	nl	ANFIS	DT	ANN(1)	ANN(2)	ANN(3)	KNN	SVM
ANFIS	0	–	↑	↑	↑	↑	↑	↑
	10	–	↑	↑	↑	↑	↑	↑
	30	–	↑	↑	↑	↑	↑	↑
	50	–	↑	–	–	–	↑	↑
DT	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	←	–
	30	←	–	←	←	←	←	←
	50	←	–	←	←	←	←	←
ANN(1)	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	–	–
	30	←	↑	–	–	–	←	–
	50	–	↑	–	–	–	–	–
ANN(2)	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	←	–
	30	←	↑	–	–	–	←	–
	50	–	↑	–	–	–	–	–
ANN(3)	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	–	↑
	30	←	↑	–	–	–	←	↑
	50	–	↑	–	–	–	–	–
KNN	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	↑
	30	←	↑	↑	↑	↑	–	↑
	50	←	↑	–	–	–	–	↑
SVM	0	←	–	–	–	–	–	–
	10	←	–	–	–	←	←	–
	30	←	↑	–	–	←	←	–
	50	←	↑	–	–	–	←	–

attribute noise.

Table 4.7: Test accuracy and RLA results of classifiers tested with noisy datasets.

	Noise Level (%)	ANFIS	DT	ANN(1)	ANN(2)	ANN(3)	KNN	SVM
Test accuracy results	0	99.271	99.986	100.000	100.000	100.000	100.000	100.000
	10	72.214	85.693	83.570	86.131	84.031	84.093	81.79
	30	41.343	63.114	59.464	63.860	60.467	59.971	55.403
	50	27.414	47.264	45.381	48.564	46.176	44.764	40.799
RLA values results	0	-	-	-	-	-	-	-
	10	0.273	0.143	0.164	0.139	0.160	0.159	0.189
	30	0.584	0.369	0.405	0.361	0.395	0.400	0.446
	50	0.724	0.527	0.546	0.514	0.538	0.552	0.592

As evinced in Table 4.7, for all classifiers, accuracy decreases exponentially with an increase in the noise level of the testing dataset. In this situation, the most robust classifiers are ANN(2), DT, ANN(3), ANN(1), and KNN.

Fig. 4.7 illustrates the results of the classification algorithms when both accuracy and testing time are considered with both clean and noise datasets (levels of 10%, 30%, and 50%). A marker identifies each classification algorithm, and a different color identifies each noise level. The light-blue bands indicate the areas of the greatest accuracy rates or the slowest testing times, and the light-purple intersection of these bands indicates the area where the best results were obtained when considering both accuracy and testing time.

The DT algorithm takes less than 30 milliseconds for classification for with the greatest

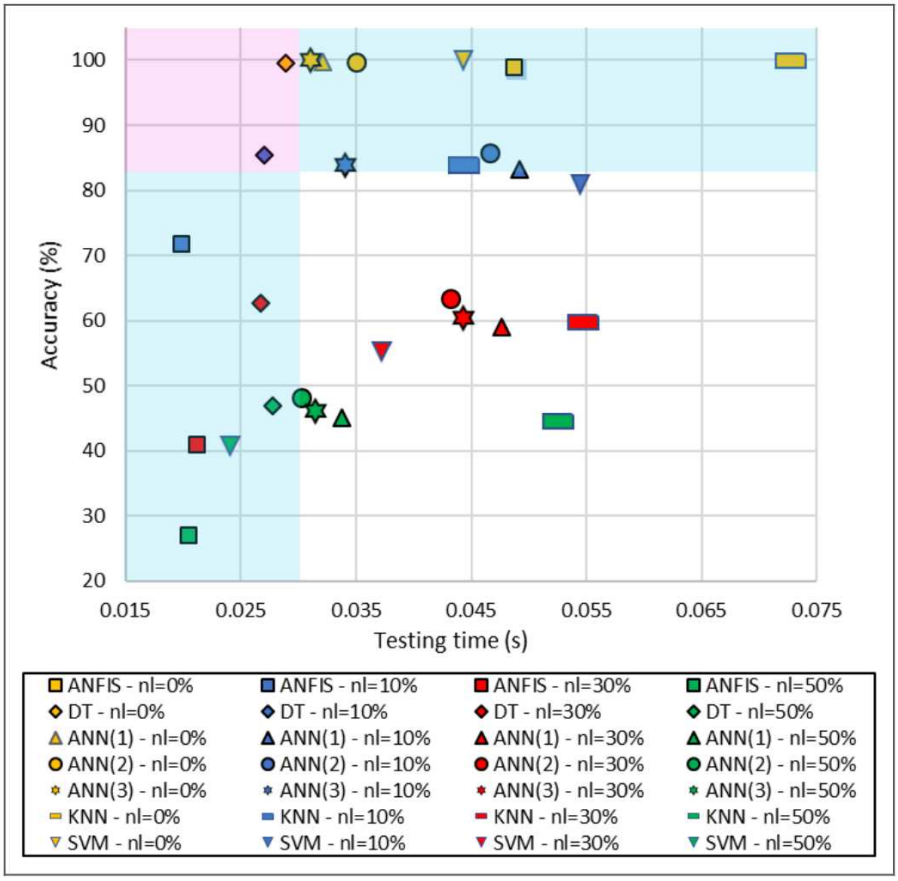


Figure 4.7: Accuracy rates concerning the testing time for classifiers tested with both clean and noisy datasets.

level of accuracy, up to 1,400 applications simultaneously arriving at the edge, when the input is a noise dataset as long as the noise level does not exceed 10%.

Table 4.8 presents the results of a two-tailed Wilcoxon signed-rank tests (considering a significance level of 0.05) to verify the statistical differences between the accuracy of the different classifiers when tested with noisy datasets. Each cell shows the result of the statistical test between the pairs of classifiers with different percentages of noise (nl). Left ' $\leftarrow$' and up '$\uparrow$' arrows indicate the greatest accuracy while an empty cell refers to "no statistical difference between the pair of classifiers" in that row and column.

The effect of noisy data set in testing is shown in Table 4.8. The performance of ANFIS is the least accurate. Moreover, the DT produces the most accurate classification results in classifications independent of the presence of noise.

4.3.5 Classification model

The final step of the proposed methodology is the selection of a classifier. The results indicate that the DT was the most accurate and robust classifier.

The decision tree algorithm does not need to assess all the attributes to classify an application since various services have exclusive features. For example, mission-critical applications are the only ones for which reliability takes on a "critical" value. Therefore, assessing certain features makes classification a more efficient process. This is an attrac-

Table 4.8: Statistical test for the accuracy of classifiers tested with noisy datasets. nl denotes the percentage of noise level present in the testing dataset.

	nl	ANFIS	DT	ANN(1)	ANN(2)	ANN(3)	KNN	SVM
ANFIS	0	–	↑	↑	↑	↑	↑	↑
	10	–	↑	↑	↑	↑	↑	↑
	30	–	↑	↑	↑	↑	↑	↑
	50	–	↑	↑	↑	↑	↑	↑
DT	0	←	–	–	–	–	–	–
	10	←	–	←	–	←	←	←
	30	←	–	←	–	←	←	←
	50	←	–	–	–	–	←	←
ANN(1)	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	←
	30	←	↑	–	↑	–	–	←
	50	←	–	–	↑	–	–	←
ANN(2)	0	←	–	–	–	–	–	–
	10	←	–	←	–	←	←	←
	30	←	–	←	–	←	←	←
	50	←	–	←	–	–	←	←
ANN(3)	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	←
	30	←	↑	–	↑	–	–	←
	50	←	–	–	–	–	–	←
KNN	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	←
	30	←	↑	–	↑	–	–	←
	50	←	↑	–	↑	–	–	←
SVM	0	←	–	–	–	–	–	–
	10	←	↑	↑	↑	↑	↑	–
	30	←	↑	↑	↑	↑	↑	–
	50	←	↑	↑	↑	↑	↑	–

tive characteristic which makes Decision Tree an ideal algorithm for the classification of applications in Fog computing. Moreover, the Decision Tree algorithm is easy to interpret, fast for fitting and prediction, and does not use much memory. Given these characteristics, the Decision Tree algorithm can be run by devices such as routers, switches, and servers, located on the first Fog layer of the reference architecture introduced by the OpenFog Consortium in [61]. After classification, the output of the classifier serves as input for the scheduler, also located at the first Fog layer, which decides where the application should be processed.

Chapter 5

Single-objective QoS-aware task scheduling for the cloud-fog continuum

Several problems in IoT and cloud-fog networks are optimization problems in essence. This is the case of task scheduling. When an application reaches the edge of the network, the tasks that compose this application need to be scheduled on processing elements along the Cloud to Things (C2T) continuum. In this regard, this chapter introduces task scheduling algorithms to support multi-class services in cloud-fog computing systems, considering the CoS of applications to decide where tasks should be processed. We first overview related work (Section 5.1). We, then, describe the cloud-fog model adopted in this study (Section 5.2). After, we detail the two proposed workflow schedulers (Section 5.3). Finally, we explain the experiments conducted to evaluate the performance of the proposed schedulers (Section 5.4).

5.1 Literature review

A fundamental question in cloud-fog systems is: where the tasks of applications should be executed: on the fog or in the cloud? Diverse studies have considered different combinations of processing elements: fog nodes [1, 2, 44, 62, 81], fog nodes and cloud [6, 25, 64], and devices and fog nodes [80]. Table 5.1 compares all these approaches, and next, we briefly describe each of them.

Oueis et al. [62] proposed a task scheduling strategy that considers the formation of clusters of fog nodes and load balancing in a cellular network. The strategy allocates computational resources at the serving cell according to an ordered list of requests and organizes clusters to satisfy the requests not yet served. Results show that users' satisfaction increases when compared to static clustering.

Intharawijitr et al. [44] proposed three different policies for mapping tasks onto fog nodes to minimize blocking while respecting latency constraints. The first policy randomly selects a fog node to host a task. The second policy selects the fog nodes to minimize the total latency, and the third one chooses a fog node to maximize resource utilization. Results show that minimum blocking is achieved by the policy that prioritizes latency as the criterion for mapping tasks.

Table 5.1: Characteristics of solutions for task scheduling in Fog systems. In the approach column, A, H, P and ILP mean analytical, heuristic, policy and integer linear programming, respectively.

Paper	Network Domain	Major contribution	Approach	Heterogeneous Resources
Oueis et al. [62]	Fog	Management of the execution of tasks in a cellular network.	H	✓
Intharawijitr et al. [44]	Fog	Three policies to select fog nodes for the execution of tasks.	P	
Aazam and Huh [1]	Fog	A loyalty-based task scheduling algorithm.	H	✓
Aazam and Huh [2]	Fog	A loyalty-based task scheduling algorithm considering different types of devices.	H	✓
Zhang et al. [81]	Fog	Fair task offloading among fog nodes in 5G fog network.	A	✓
Agarwal et al. [6]	Cloud-Fog	Workload distribution for the reduction of the response time and cost.	H	✓
Deng et al. [25]	Cloud-Fog	Workload distribution to minimize power.	H	✓
Pham and Huh [64]	Cloud-Fog	A good trade-off between makespan and the cost of task execution	H	✓
Zeng et al. [80]	IoT-Fog	Minimization of the completion time by jointly considering task scheduling and image placement.	H	✓
This thesis	Cloud-Fog	Minimize of the completion time by considering the CoS of the application to decide where tasks should be processed.	ILP	✓

Aazam and Huh [1] introduced a service-oriented resource management model for fogs, which deals with the uncertainty associated with resource utilization of heterogeneous IoT devices. This model employs an estimation of resource utilization, based on Relinquish Probability. Moreover, it considers the history of resource utilization and prioritizes the allocation of resources to frequent users. Aazam and Huh [2] extend their model [1] by classifying the type of accessing device based on the nature of the device and mobility. They also introduced a pricing strategy based on an incentive mechanism for frequent users.

Zhang et al. [81] investigated the problem of fair task offloading among fog nodes in a 5G fog network. They proposed a scheduler that considers the delay and energy consumption of tasks. The authors defined a fairness index to select fog nodes to minimize task delay.

Agarwal et al. [6] presented architecture and algorithms for resource provisioning in a cloud-fog environment. The proposed architecture consists of three layers: client, fog, and cloud. In this architecture, a fog manager enlists all the processors available to the client. The fog server verifies whether sufficient computational resources are available on the fog layer. Depending on the resource availability, tasks can be fully or partially executed on the fog layer or sent to the cloud. Results show that the proposed algorithm improves the response time, bandwidth utilization, and power consumption.

Deng et al. [25] employed optimization to study the trade-off between power consumption and delay in a cloud-fog system. They modeled the power consumption and delay of each element of the cloud-fog system in a resource allocation problem. Convex optimization was employed to minimize the energy consumed by a fog layer for a particular input workload, while a heuristic was employed to optimize the energy consumption of a cloud. Moreover, the authors attempted to optimize the energy consumption of communications. Results show that, by sacrificing few computation resources, it is possible to save bandwidth and reduce delay.

Pham and Huh [64] proposed a task scheduler to maximize the efficiency of large-scale offloading applications. The aim is to achieve a trade-off between makespan and monetary cost. The scheduler sorts the tasks based on the length of the critical paths from each task to the final task and then select the adequate node to execute the tasks. To calculate the monetary cost, the authors considered that the fog provider rents both virtual host and network bandwidth from cloud providers to extend the capabilities of the fog nodes. To prove that the proposed algorithm can achieve better tradeoff value between the makespan and the cost of task execution than other methods, authors defined a comparison criteria called Cost Makespan Tradeoff (CMT). Compared with Greedy for Cost algorithm, which achieves the minimum monetary cost but long schedule length, the proposed algorithm had better CMT in all cases. The HEFT algorithm achieved the minimum schedule length but it went with the significant increase of cost. The Dynamic Level Scheduling (DLS) algorithm also achieved small schedule length, but it requires much more cost for cloud resources and thus gets the worst CMT value, which is about from 15% to 25% lower than the proposed algorithm.

Zeng et al. [80] proposed a formulation to decide whether the processing of tasks should be carried out on client devices or at edge nodes. The aim is to minimize the completion

time by jointly considering task scheduling and virtual machine image placement using a mixed-integer nonlinear programming problem. The completion time includes the computation time, I/O time, and transmission time. Results show that the proposed algorithm outperforms greedy solutions for different task arrival rates, client processing rates, server processing rates, and disk I/O rates.

In this chapter, we introduce two task scheduling algorithms to support multi-class services in cloud-fog computing systems. None of the previous proposals for scheduling tasks in cloud-fog systems [1, 2, 6, 25, 44, 62, 64, 80, 81] considered the CoS of applications to decide where tasks should be processed. Besides that, the proposed scheduling algorithms take into consideration different types of resources, such as CPU, RAM, and storage space. The two proposed schedulers are based on integer linear programming formulation. The first, called CASSIA-INT, solves an exact integer programming formulation, while the second, named CASSIA-RR, solves a relaxed version of the integer programming formulation.

5.2 System model

This section introduces models for the infrastructure and application representation as well as provides scenarios for the proposed system model.

5.2.1 Infrastructure and application models

In this section, it is assumed a cloud-fog system based on the architecture presented in Figure 3.1, which is composed of a fog stratum with two layers and one cloud [28]. Additionally, there is a set of isolated fog nodes intended to process mission-critical applications. Processing elements on this cloud-fog system have computational, memory, and storage resources.

The end-user submits an application to be processed at the network edge. An application consist of a workflow and QoS requirements. The QoS requirements are employed by a classifier to identify the CoS associated with the application. The CoS can be Mission-critical (MC), Real-time (RT), Interactive (IN), Conversational (CO), Streaming (ST), CPU-bound (CB), or Best-effort (BE), in accordance to the criteria defined in Table Table 3.2. The CoS label is used by the scheduler to decide the layer the tasks of an application should be scheduled.

The scheduler considers the workflow and the label of an application as well as the cloud-fog current resource availability to make scheduling decisions, as illustrated in Fig. 5.1.

An application workflow is composed of dependent tasks. Dependent tasks have a topological order which must be respected by the schedule, and their relations are represented by a Directed Acyclic Graph (DAG) [12, 15]. In a $DAG = (V_D, E_D)$, the set of vertices $V_D = \{i, j, ..., m\}$ denotes the set of subsequent tasks, and on edge $\{ij\} \in E_D$ represents the precedence constraint such that a task i should complete its execution before a task j starts [74]. Fig. 5.2 illustrates the DAG of an application workflow. Each node of the DAG represents a task $j \in V_D$. The three values in the labels represent the

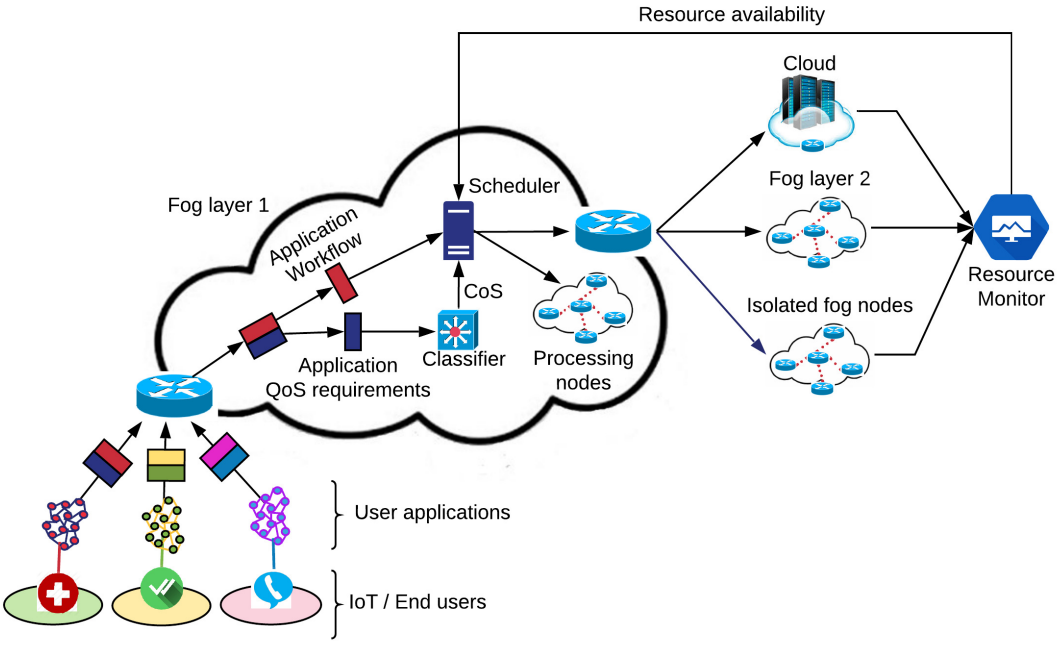


Figure 5.1: Task scheduling in the cloud-fog system.

number of instructions to be processed (I_j), the amount of RAM demanded (M_j), and the storage required to process the task, (S_j).

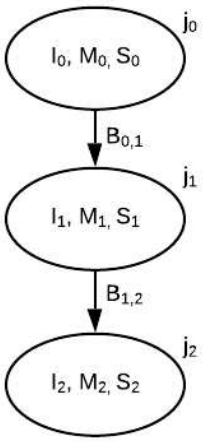


Figure 5.2: DAG of the application workflow.

Likewise, an edge $\{ij\}$ has a non-negative weight value B_{ij} that represents the number of bytes transmitted from task i to task j. Thus, a task cannot begin execution until all its inputs have been received. Moreover, we assume without loss of generalization that the DAG has a single entry task (no preceding task) and a single exit task (no succeeding task).

The cloud-fog topology is represented by a graph $H = (V_H, E_H)$, where V_H is the set of vertices and contains the processing elements on different layers of the cloud-fog system. These processing elements can be isolated fog nodes (IS), nodes at the first fog layer (F1), nodes at the second fog layer (F2), nodes at the first and second fog layer and

in the cloud (FC); and nodes exclusively in the cloud (CL). Thus, $V_H = \{(IS) \cup (F1) \cup (F2) \cup (FC) \cup (CL)\}$.

Fig. 5.3 illustrates a topology graph of the cloud-fog system. Nodes represent processing elements with labels expressing the inverse of the processing capacity, (TI_k), in (instructions/units of time)$^{-1}$, the available RAM, (R_k), and the available storage (D_k). The edges represent network links, with labels indicating the inverse of the available bandwidth, (TB_k), in (bits/units of time)$^{-1}$. The dotted lines suggest that there may be more than one processing element at each processing layer.

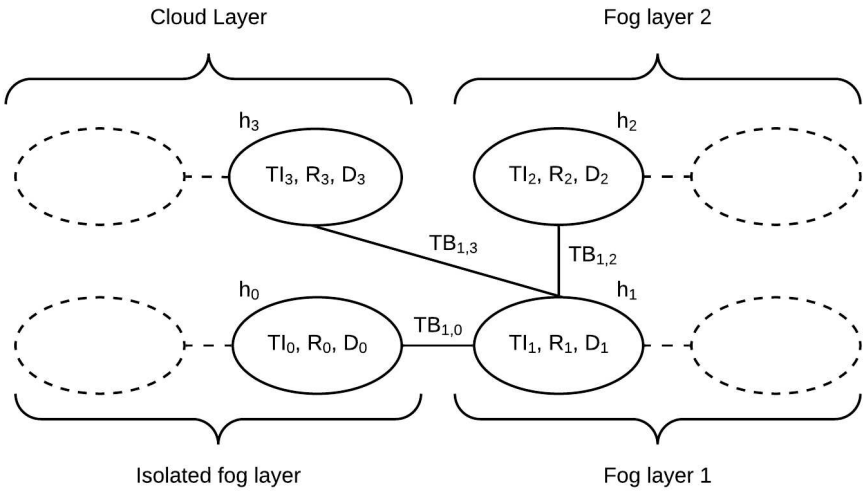


Figure 5.3: Topology graph of the cloud-fog network.

A subgraph contained in the cloud-fog network topology is denoted as V_H' and represents the processing layer on which an application should be processed according to its CoS. Table 3.3 shows the relationship between the CoS of the fog computing applications and the processing layers of the cloud-fog topology.

5.2.2 Application scenarios

The integration of cloud and fog processing resources offers final users the possibility of executing both real-time and non-delay constraint applications. In this section, the behavior of these two kinds of applications is analyzed, while a scheduling strategy based on CoS determines the execution sequence of the tasks. For this purpose, the EEG tractor beam game (EEGTBG) application and a video surveillance service (VSOT) were used in the study. The operation of the modules that compose both applications, as well as their QoS requirements and resource demands, are explained below.

In the EEG tractor beam game (EEGTBG) application, each player needs to wear a wireless EEG headset that is connected to his/her smartphone. The game runs as an application on a user's smartphone. On the display of the application, the 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 (estimated using a ratio of the average power spectral density in the EEG α, β and θ bands of the player) [35].

To win the game, a player should try to pull the target toward himself by exercising concentration while depriving other players of their chances to grab the target.

Fast processing and low response times must be guaranteed to provide a high quality of experience (QoE). The EEGTBG application is composed of five modules (Fig. 5.4): EEG sensor, display, client, concentration calculator, and coordinator. We have considered an application model consisting of three major modules, hereinafter, referred to as EEGTBG tasks: client, concentration calculator, and coordinator. The client receives raw EEG signals from the sensor and filters consistent data to the concentration calculator, which computes the user's concentration level. On receiving the concentration level, the client sends the value to the display. The concentration calculator computes the concentration level of the user. The coordinator sends the current status of the game to all players. The CPU demands of the EEGTBG tasks are described in Table 5.2.

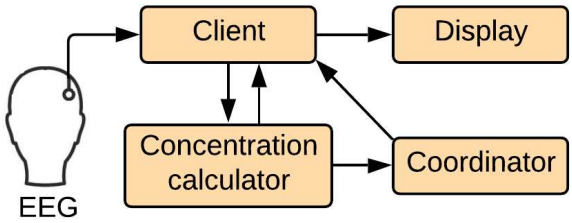


Figure 5.4: Modules of the EEGTBG application.

Table 5.2: CPU demands estimated for each task of the EEGTBG application [13].

Task	Client	Concentration calculator	Coordinator
MIPS	200	350	100

Moreover, each task of the EEGTBG application demands 0.5 GB of RAM and 3 GB of storage space. Additionally, 300 MB of data is exchanged between tasks.

The VSOT application relies on a set of distributed intelligent cameras that can track movement. Hence, the QoE experimented by the user depends on the available processing resources. Sufficient processing power should be guaranteed for a good experience. The VSOT application consists of five modules (Fig. 5.5): motion detector, object detector, object tracker, user interface, and the control of pan, tilt, and zoom (PTZ). The motion detector module filters the raw video streams captured by the camera and forwards the frames of the video to the object detector module. The object detector module identifies the moving objects and sends object identification and position information to the object tracker. The object tracker receives the coordinates of the tracked objects, computes an optimal PTZ configuration of all the cameras covering the area, and sends a command to the PTZ control module. The PTZ module, located in each smart camera, adjusts the physical camera according to the PTZ parameters sent by the object tracker module. Finally, the user interface module sends a fraction of the video streams containing each

tracked object to the user's device. Since it is assumed that the PTZ control module is always placed in the camera, our application model of the VSOT application is composed of four modules, from now on, referred to as the VSOT tasks: motion detector, object detector, object tracker, and user interface. The CPU demands of tasks are described in Table 5.3.

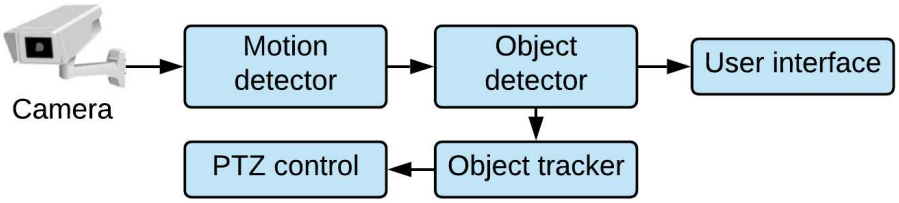


Figure 5.5: Modules of the VSOT application.

Table 5.3: CPU demands estimated for each task of the VSOT application [13].

Task	Motion detector	Object detector	Object tracker	User interface
MIPS	300	550	300	200

Each task of the VSOT application demands 1 GB of RAM and 7 GB of storage space. Additionally, 500 MB of data is exchanged between tasks.

The EEGTBG and the VSOT applications have special characteristics that emphasize the effect of using CoS in the scheduling of application workflows. By using the application classification in [33], the EEGTBG and VSOT applications are labeled as real-time and CPU-Bound applications, respectively. An EEGTBG application must be processed as close as possible to the end-user, i.e., at the first fog layer, while a VSOT application can be processed at any layer of the cloud-fog system.

5.3 Proposed scheduling approaches

This section introduces the CASSIA-INT scheduler, which is based on an integer linear programming formulation, as well as the CASSIA-RR scheduler that implements the randomized rounding technique. The solution given by the randomized rounding algorithm is an approximation to the exact solution given by CASSIA-INT.

The problem formulation considers that tasks composing the application are executed one at a time. It is also assumed that when a task arrives at the processing host, the virtual machines (VMs), where tasks will run, are already activated.

The schedulers receive as input information regarding the availability of fog and cloud resources, including processing elements and network links; the CoS of the application, assigned by a classifier, and a DAG describing the workflow containing processing, RAM, storage, and communication demands of the tasks that make up the fog application.

The schedulers are based on an integer linear program designed to minimize the makespan of a schedule. The following notation is used to represent an application composed of tasks and dependencies:

- m: number of tasks that make up the application ($m \in \mathbb{N}$),
- I_j: processing demand of j^{th} task, expressed as the number of instructions to be processed ($I_j \in \mathbb{R}_+$),
- M_j: RAM demand of j^{th} task ($M_j \in \mathbb{R}_+$),
- S_j: storage demand of j^{th} task, expressed as the number of data units required by the task ($S_j \in \mathbb{R}_+$),
- $B_{i,j}$: number of data units transmitted between the i task and the j task ($B_{i,j} \in \mathbb{R}_+$),
- V_D: set of vertices representing the tasks of the application workflow,
- E_D: set of edges representing the dependencies among tasks $\{ij : i < j\}$.

The cloud-fog topology is represented by the following notation:

- n: number of processing elements of the cloud-fog system,
- o: task scheduler,
- R_k: available RAM capacity of the k^{th} host,
- D_k: available storage capacity of the k^{th} host,
- k_r: fastest processing node ($k_r \in V_H^{'}$),
- $\delta(k)$: set of processing elements linked to the k^{th} processing element in the network, including the processing element k,
- $TB_{k,h}$: time for transmitting a data unit on the link connecting the k^{th} processing element and the h^{th} processing element ($TB_{k,h} \in \mathbb{R}_+$),
- TS_k: time for transmitting a data unit on the link connecting the scheduler and the host k ($TS_k \in \mathbb{R}_+$),
- TI_k: the time the k^{th} host takes to execute a single instruction ($TI_k \in \mathbb{R}_+$),
- V_H: set of network hosts,
- $V_H^{'}$: subset of network hosts defined by the CoS of the application,
- E_H: set of network links between hosts.

Moreover, three parameters are computed for each task. The first parameter, $P_{j,k}$, is the processing time of the task j on the host k, expressed as

$$P_{j,k} = I_j TI_k$$

The second parameter, C_{ij}, is the time taken for data transfer between two subsequent tasks i and j of the DAG, hosted, respectively, on the processing elements h and k.

$$C_{ij} = B_{ij} TB_{hk}$$

The third parameter, $L_{j,k}$, represents the sum of the transmission delay, $d_{trans_{j,k}}$, and the propagation delay, $d_{prop_{j,k}}$. The transmission delay of all data of a task is computed by

$$d_{trans_{j,k}} = S_j TS_k$$

The propagation delay between the scheduler o and the host k is the distance between the two hosts divided by the propagation speed of the link, ω.

$$d_{prop_{o,k}} = \frac{d_{o,k}}{\omega}$$

Both transmission and propagation delays are considered from nodes at fog layer 1 to nodes at fog layer 1, fog layer 2, on the cloud, or to processing elements in isolated hosts.

Moreover, time is discretized and defined as $T = \{1, ..., T_{max}\}$, where T_{max} is the time that the fastest host in $V_H^{'}$ would take to execute the tasks of the application serially. T_{max} is calculated by

$$T_{max} = TS_{k_r} \sum_{j=0}^{m-1} S_j + m\left(d_{prop_{o,k_r}}\right) + TI_{k_r} \sum_{j=0}^{m-1} I_j + \max\left[B_{i,j} TB_{h,k}\right](m-1);$$

where,

$$TI_{k_r} = min\left(TI_{k|k \in V_H^{'}}\right)$$

The solution for the integer linear program is given by the value of the decision variable $x_{j,t,k}$ which is 1 if the j^{th} task should be processed at time t on the host k. The integer linear programming problem is formulated as follows:

Minimize

$$C_{max}$$

Subject to

$$\sum_{t \in T} \sum_{k \in V_H^{'}} x_{j,t,k} = 1, \quad \forall j \in V_D; \tag{5.1}$$

$$\sum_{t \in T}(t + L_{j,k} + P_{j,k})x_{j,t,k} \leqslant C_{max}, \quad \forall j \in V_D, k \in V_H^{'}; \tag{5.2}$$

$$\sum_{j \in V_D} \sum_{s=t}^{t+P_{j,k}-1} x_{j,s,k} \leqslant 1, \quad \forall k \in V_H^{'}, t \in T, \text{ where } t \leqslant T_{max} - P_{j,k}; \tag{5.3}$$

$$\sum_{s=1}^{t} x_{j,s,k} \leqslant \sum_{h \in \delta(k)} \sum_{s=1}^{t-L_{i,h}-P_{i,h}-C_{ij}} x_{i,s,h}, \quad \forall j \in V_D, k \in V_H^{'}, ij \in E_D, t \in T; \tag{5.4}$$

$$\sum_{t \in T} \sum_{j \in V_D} M_j x_{j,t,k} \leqslant R_k, \quad \forall k \in V_H^{'}; \tag{5.5}$$

$$\sum_{t \in T} \sum_{j \in V_D} S_j x_{j,t,k} \leqslant D_k, \quad \forall k \in V_H^{'}; \tag{5.6}$$

$$x_{j,t,k} \in \{0, 1\}, \quad \forall j \in V_D, k \in V_H^{'}, t \in T; \tag{5.7}$$

$$R_k \geqslant 0, \quad \forall k \in V_H^{'}; \tag{5.8}$$

$$D_k \geqslant 0, \quad \forall k \in V_H^{'}; \tag{5.9}$$

The objective is to find a schedule for V_D on $V_H^{'}$ that minimizes the makespan (C_{max}), with ending time equals to the completion time of the last task in V_D. Constraint (5.1) establishes that each task (j) must be executed at once in a single host k. Constraint (5.2) ensures that the makespan is at least the longest completion time of the last operation of all tasks. Constraint (5.3) establishes that there is at most one task in execution on any host at any given time. Constraint (5.4) establishes that the execution of the j^{th} task cannot start until all its predecessor tasks have been completed and the data required by the j^{th} task transferred. Constraint (5.5) determines that the total amount of RAM occupied by the tasks processed on host k should not exceed its available RAM capacity. Constraint (5.6) establishes that the total amount of data occupied by tasks processed on host k should not exceed its available storage space capacity. Constraint (5.7) defines the domain for the variable $x_{j,t,k}$. Constraints (5.8) and (5.9) state that the variables R_k and D_k can only take non-negative integer values.

The CASSIA-INT implements the integer linear program described above, while the

CASSIA-RR applies the randomized rounding relaxation technique to reduce the execution time to produce a solution without significant loss of the quality of the schedule.

5.3.1 Task scheduling based on Integer linear programming

The CASSIA-INT scheduler is presented in Algorithm 5.1. The integer linear program returns an exact schedule that gives the mapping of tasks on hosts. Integer programming is classified as an NP-complete problem. In addition, the time taken by the optimizer to solve the ILP problem dominates the computational complexity of the CASSIA-INT scheduler.

Algorithm 5.1: CASSIA-INT Scheduler

Input : ILP:Integer linear program formulation.
Output: Schedule V_D on $V_H^{'}$ along with the schedule of the necessary hosts $k \in V_H^{'}$.

1 Solve the ILP.
2 Let X be the solution schedule V_D on $V_H^{'}$, where $X = X_{j.k}$ and $X_{j.k} = \sum_{t \in T} x_{j,t,k}, \forall j \in V_D, k \in V_H^{'}$
3 Return the schedule.

5.3.2 Task scheduling based on approximation algorithms

The scheduling problem is an NP-Complete problem [65]. Techniques such as approximation algorithms provide solutions that guarantee solutions close to the optimum [14]. The CASSIA-RR scheduler implements an approximation algorithm called *Randomized Rounding*, presented in Algorithm 5.2. First, the integer linear program is executed with the relaxation of the decision variables to turn the integer program into a linear program. The relaxation of the discrete time formulation consists of replacing the set {0,1} in constraint (7) by the interval [0,1]. The values found when solving this linear formulation represent the probability values of each task to be executed on a host. Algorithm 5.2 is executed P times, and the schedule with the shortest makespan is chosen. P must be defined in order to increase the chances of obtaining a good schedule. The execution time of the CASSIA-RR scheduler increases with the value of P. Algorithm 5.2 runs in $O(\alpha_{\mathscr{P}} + P \cdot (|V_D| \cdot |V_H^{'}| + |E_D| + |V_H^{'}|))$, where $\alpha_{\mathscr{P}}$ represents the time complexity to set each variable of the linear program, $X_{j,k}$, which is at least $O(|V_D| \cdot |V_H^{'}|)$.

5.4 Performance Evaluation

This section assesses the performance of the CASSIA-INT and CASSIA-RR schedulers. The cloud and fog processing layers were represented by graphs that describe the topology of the network. Vertices of the graphs represent processing elements, and the edges represent communication links. The topologies of the graphs are given by the Barabási-Albert model [11], a method for generating network topologies similar to those formed

Algorithm 5.2: CASSIA-RR Scheduler

Input : ILP: Integer linear program formulation.
P: Number of drawings.
Output: Schedule of V_D in $V_H^{'}$ along with the schedule of the necessary hosts $k \in V_H^{'}$.

```
Execute the ILP relaxed as a linear program (LP).
Let X be the solution schedule of V_D in V_H' for the LP, where X = X_{j,k}
for P times do
    for each task j ∈ V_D do
        Let X_{j,k} be the probability of mapping the task j on the host k, select a
         host at random where the task j should be executed, based on the
         previous mapping probability.
        Add the task selected to a provisional list of tasks to be executed.
    if the provisional list meets the precedent constraint then
        Add the provisional list to the final list of schedules if it doesn't already
         exist
Keep the provisional list in the final list producing the shortest schedule.
Return the schedule.
```

by resources on the Internet. The Barabási-Albert model was implemented using the topology generator BRITE [58]. Table 5.4 shows the parameters of the configuration file passed to BRITE.

Table 5.4: BRITE configuration parameters used to generate the network topology.

Parameter	Meaning	Value
HS	Size of one side of the plane	1000
LS	Size of one side of a high-level square	1000
N	Number of nodes	int $1 \leq N \leq HS * HS$
Node placement	Heavy Tailed	2
m	Number of neighboring node each new node connects to.	1
BWDist	Bandwidth Distribution	Uniform
BWMin	Minimum bandwidth value	1 Gbps
BWMax	Maximum bandwidth value	10 Gbps

Experiments conducted consider two types of applications: the electroencephalography (EEG) tractor beam game (real-time) and a video surveillance/object tracking (VSOT) application (delay tolerant) [35]. The operation of the modules that compose these applications is explained in detail in Section 5.2.2.

Six different scenarios were used by varying the CoS of the application, the number of hosts in the network, and the network load to study the operation of the schedulers. The first three scenarios implement the EEGTBG application, while the remaining three

implement the VSOT application. The number of hosts was 36, 108, and 180, and the mean resource and bandwidth utilization were 0%, 20%, 40%, and 60%. The utilization level of each node and link were randomly determined so that an average target utilization could be achieved.

Different values of P were tested for the CASSIA-RR; a value of 10,000 is recommended since no further improvement in results can be obtained by using a larger value.

Furthermore, links with 2 Gbps, 1 Gbps, and 500 Mbps of bandwidth connect the scheduler with the fog layer 1, the fog layer 2, and the cloud, respectively. Two other 10 Gbps links connect the fog layer 1 to the fog layer 2, as well as connect the fog layer 2 to the cloud.

The schedulers were implemented in Java language using the IBM ILOG CPLEX Optimization Studio V12.6.0 to solve the ILP model and its relaxation. All the programs were executed on an AWS R5 instance Memory Optimized Intel Xeon 3.1 GHz EBS-Only with 20 GiB storage, 128 GB of RAM, and CentOS 7.0 operating system.

The processing, RAM memory, and storage capacity of each type of host in the network are shown in Table 5.5.

Table 5.5: Characteristics of the processing hosts distributed along the processing layers of the architectural model.

Device's Location	**Processing Capacity (MIPS)**	**RAM (GB)**	**Storage (GB)**
Fog layer 1	1400	2.8	28
Fog layer 2 / Isolated fog nodes	2800	4.2	42
Cloud	3600	8.4	84

To assess the performance of the proposed algorithms, we consider two traditional scheduling schemes, namely Random and Round Robin, as the baseline algorithms.

By definition, our proposed algorithms choose the more appropriate processing layer to execute a task according to the QoS requirements of an application. In contrast, Random and Round Robin algorithms allocate tasks for processing at any layer of the architecture, which may result in a QoS violation. We ascertain a QoS violation by examining both the QoS requirements of the application and the layer where the task is processed. If the processing layer chosen by the scheduling algorithm to process the application differs from the potential layers for its CoS, the scheduling algorithm incurs in QoS violation. An allowed processing layer guarantees the support of the QoS requirements of the applications. Table 5.6 shows the results of the analysis of violation of QoS requirements for the Random and Round Robin algorithms.

As can be seen in Table 5.6, the percentages of QoS violations for the EEGTBG application are higher than they are for the VSOT application. VSOT can be processed at the fog layer 1, fog layer 2, and the cloud, while EEGTBG can only be processed at the fog layer 1 since it is a delay-sensitive application. Hence, the probability of choosing a layer not allowed is higher for EEGTBG than it is for VSOT.

Table 5.6: Percentage of QoS violations for different values of network load: 0%, 20%, 40%, and 60%. S1:n=36, S2:n=108, and S3:n=180.

Algorithm	Network Load (%)	EEG			VSOT		
		S1	**S2**	**S3**	**S1**	**S2**	**S3**
Random	**0**	52%	60%	64%	0%	44%	34%
	20	74%	78%	64%	22%	28%	26%
	40	44%	54%	52%	0%	0%	12%
	60	70%	72%	48%	0%	0%	40%
Round Robin	**0**	72%	56%	64%	26%	24%	20%
	20	60%	70%	56%	18%	24%	24%
	40	52%	48%	46%	20%	22%	12%
	60	46%	48%	54%	22%	26%	26%

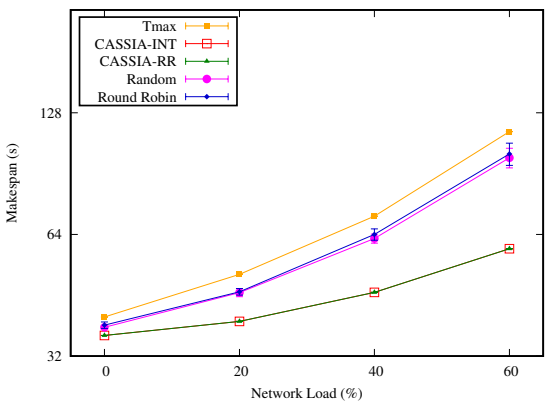
Figure 5.6 and Figure 5.7 show the makespan and the execution time of the EEGTBG and the VSOT applications as a function of the network load. The points in the figures correspond to mean values with a 95% confidence interval. In the makespan graphics, the time taken to execute all tasks in sequence (Tmax) is plotted for reference.

As the percentage of network load increases, the makespan of the application also increases. The makespan values plotted in Figures 6 and 7 show that our proposed algorithms outperform the baseline algorithms. The advantage of our algorithm is the result of taking the makespan into account in the optimization. Moreover, our algorithm also avoids QoS violations (see Table 5.6).

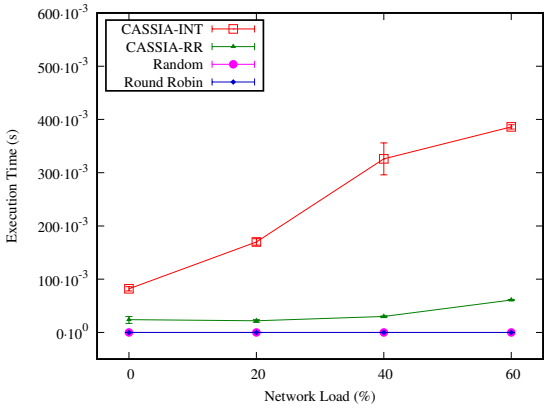
The execution time values plotted in Figures 6 and 7 show that both CASSIA-INT and CASSIA-RR schedulers need more time to generate a schedule as the number of hosts increases. However, the CASSIA-RR scheduler demands much shorter execution times than do the CASSIA-INT and yet produces similar makespan values. Although the execution time values of Random and Round Robin algorithms are shorter than those of our proposed algorithms, which produces less QoS violations than the two other algorithms.

Figures 6 and 7 show that there is a great difference between the results obtained by the EEGTBG and VSOT. This is due to the association between the CoS of the application and the processing layer on which the application is executed. While EEGTBG is a real-time application that must be processed as close as possible to the end-user, VSOT is a CPU-Bound application, requiring a much larger number of processing resources and can involve almost all layers of the reference architecture in the processing of tasks. For this reason, the makespan values of the VSOT application are greater than those of the EEGTBG application.

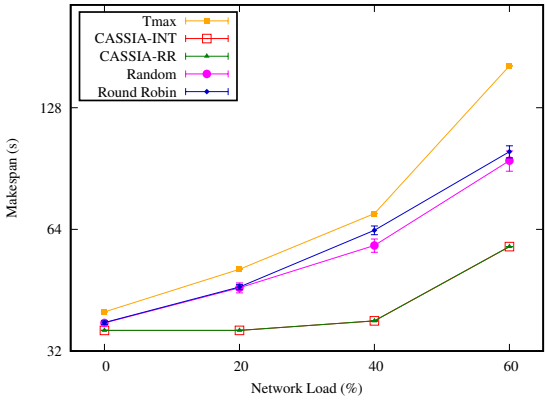
Although the makespan values produced by the CASSIA-RR and the CASSIA-INT scheduler are very close, CASSIA-RR considerably reduces the execution time to schedule the tasks of an application, even in the presence of a network load of 60%. These



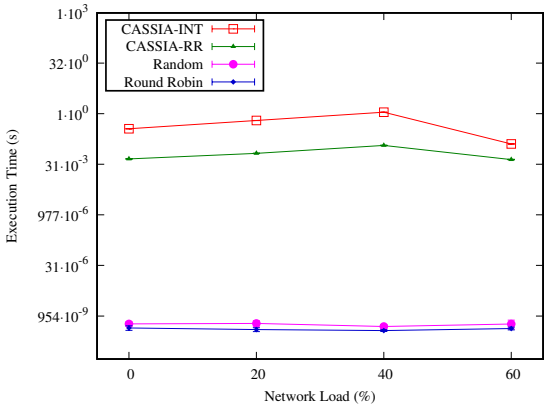
(a) S1:Makespan.



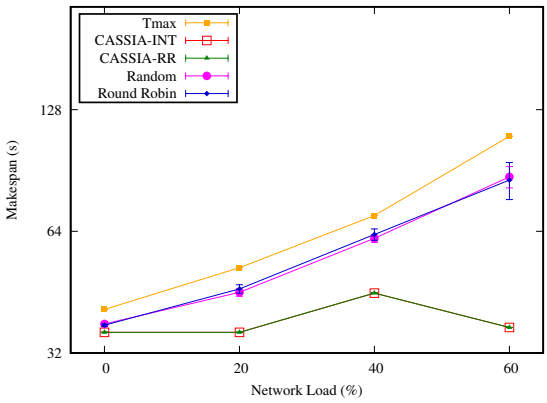
(b) S1:Execution Time.



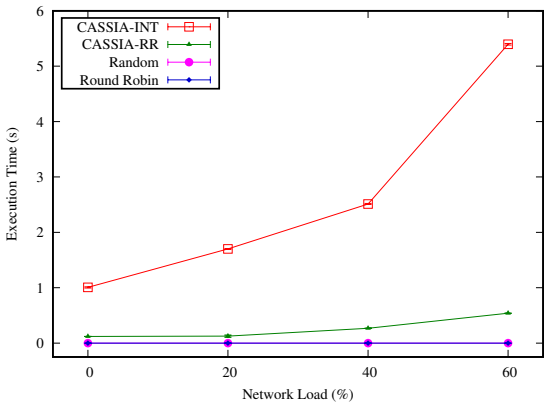
(c) S2:Makespan.



(d) S2:Execution Time.



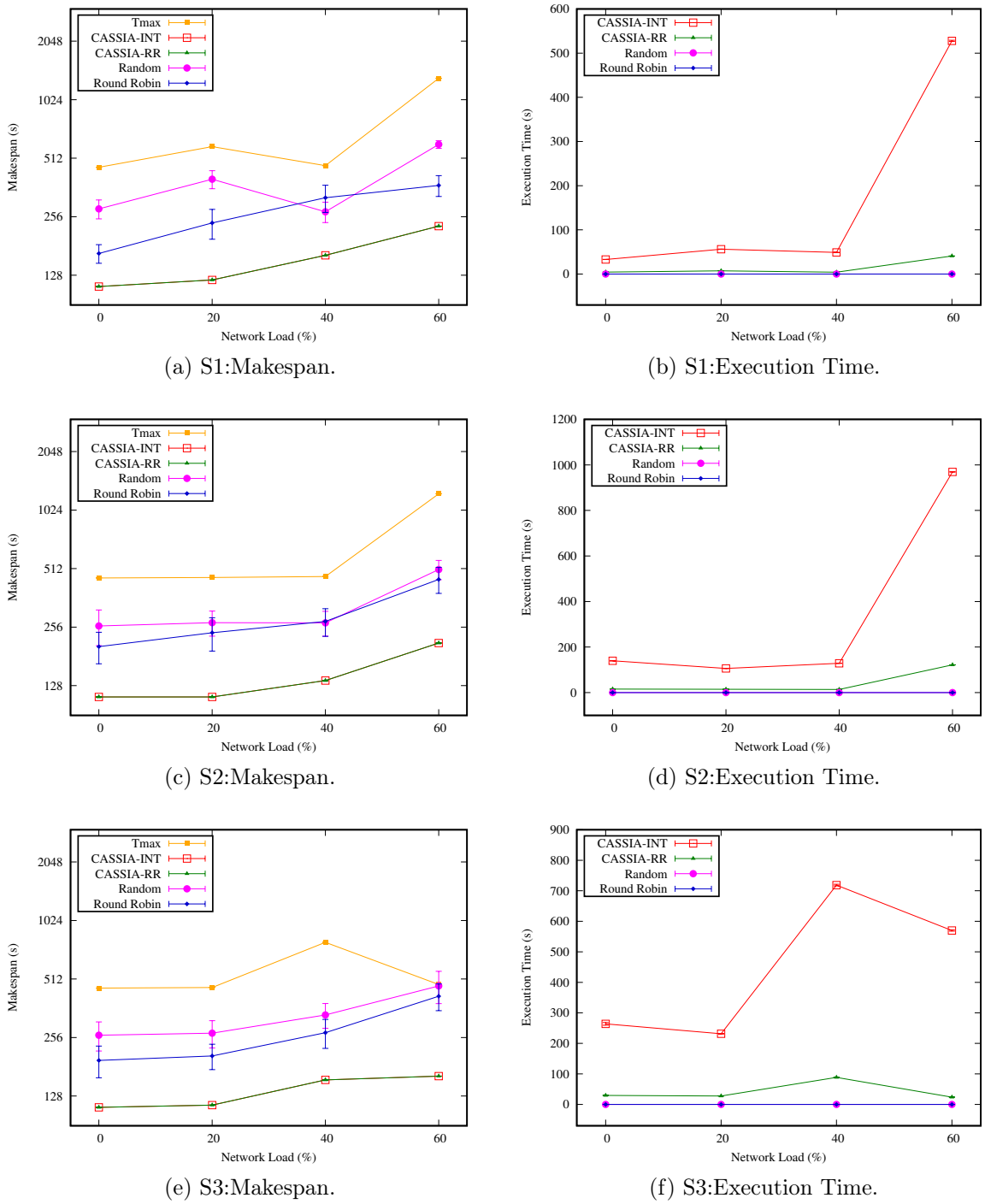
(e) S3:Makespan.



(f) S3:Execution Time.

Figure 5.6: Makespan and Execution Time of the EEGTBG application for selected scenarios. S1:n=36, S2:n=108, and S3:n=180.

characteristics make the CASSIA-RR a scheduling mechanism ideal to be implemented in environments that require short response times for the processing of applications.



(a) S1:Makespan.

(b) S1:Execution Time.

(c) S2:Makespan.

(d) S2:Execution Time.

(e) S3:Makespan.

(f) S3:Execution Time.

Figure 5.7: Makespan and Execution Time of the VSOT application for selected scenarios. S1:n=36, S2:n=108, and S3:n=180.

Chapter 6

Multi-objective QoS-aware task scheduling in the cloud-fog continuum

In the cloud-fog environment, the heterogeneity of applications' Quality of Service (QoS) requirements and the capacities of the devices impact on the scheduling decisions that ensure that applications will finish their execution at most at their required deadlines. Moreover, in task scheduling, the minimization of the makespan and processing cost constitutes conflicting objectives. It is also widely known that scheduling multi-task workflows on a distributed platform is an NP-hard problem. The problem becomes even more complicated when the dependencies of tasks have to be taken into account. To address these issues, in this chapter, we propose three multi-objective task scheduling algorithms for the cloud-fog continuum that minimize both the makespan and the processing cost of the workflow, considering the QoS requirements of the applications. The system model adopted in this chapter is the same as that adhered to in Chapter 5. We begin by reviewing relevant solutions in multi-objective task scheduling in Fog computing, Cloud computing, and the Cloud-Fog continuum (Section 6.1). We, then, present the first two multi-objective schedulers, one based on integer linear programming and the other based on approximation algorithms (Section 6.2). Next, we describe the third multi-objective scheduler, using a reinforcement learning-based algorithm (Section 6.3). Finally, we evaluate the performance of the three proposed scheduling algorithms (Section 6.4).

6.1 Literature review

Diverse studies have investigated the multi-objective task scheduling problem in different domains: cloud computing [10, 29, 46, 53, 85, 86], fog computing [5, 78], and more recently, the cloud-fog continuum [40]. Table 6.1 summarizes the previously mentioned works, which are also described below.

Table 6.1: Characteristics of multi-objective scheduling solutions in Cloud, Fog and Cloud-Fog systems.

Paper	Network Domain	Multi-objective optimization problem	Approach
Gao et al. [29]	Cloud	Optimize the makespan and economic cost.	An hybrid algorithm combining genetic algorithm, artificial bee colony optimization and decoding heuristic.
Liu et al. [53]	Mobile cloud	Optimize total cost, mean load, and the deadline-constraint meeting rate.	A heterogeneous earliest finish time (HEFT) using technique for order preference.
Alsadie [10]	Cloud	Optimize energy usage, makespan and cost.	A metaheuristic framework for dynamic virtual machine allocation using NSGA-II algorithm.
Jiahao et al. [46]	Cloud	Minimize the execution time and optimize load balancing.	A workflow scheduling algorithm based on Reinforcement Learning (Q-learning).
Zuo et al. [86]	Cloud	Optimize resource utilization according to deadline and cost constraints.	Combine two single-objective scheduling approaches with an entropy optimization model to establish a multi-objective scheduling algorithm based on ant colony optimization.
Zhu et al. [85]	Cloud	Optimize the makespan and cost.	An Evolutionary Multi-objective Optimization (EMO)-based algorithm.
Yang et al. [78]	Fog	Optimize the execution time and the resource cost.	An algorithm based on the adaptive neighborhood method.
Abdel-Basset et al. [5]	Fog	Reduce the makespan and the carbon dioxide emission rate.	An hybrid approach based on the marine predators' algorithm integrated with the polynomial mutation.

Continued on next page

Table 6.1: (Continued from previous page)

Paper	Network Domain	Multi-objective optimization problem	Approach
Ali et al. [8]	Cloud-Fog	Minimize both the makespan and total costs.	Discrete Non-dominated Sorting Genetic Algorithm II (DNSGA-II).
Hoseiny et al.[40]	Cloud-Fog	Optimize total computation time, energy consumption, and the percentage of tasks completed before their deadline.	A priority-aware genetic algorithm, PGA.
This thesis	Cloud-fog	Minimize the makespan and the processing cost.	Three task scheduling approaches based on ILP, approximative, and RL algorithms, considering the QoS requirements of the application.

Gao et al. [29] and Zhu et al. [85] aim at minimizing both the makespan and cost. To achieve this goal, authors in [29] propose a hybrid algorithm combining genetic algorithm, artificial bee colony optimization and decoding heuristic for scheduling scientific workflows over the available cloud resources, offering a limited amount of instances and a flexible combination of instance types. On the other hand, in [85], authors propose an Evolutionary Multi-objective Optimization (EMO) algorithm to solve the workflow scheduling problem on an Infrastructure as a Service (IaaS) platform.

Liu et al. [53] present HEFT-T, an algorithm that combines HEFT algorithm with TOPSIS method to minimize both total cost and mean load in mobile cloud computing.

Alsadie [10] present a metaheuristic framework called MDVMA for optimizing multiple conflicting objectives like energy usage, makespan, and cost using NSGA-II algorithm.

Jiahao et al. [46] implement a Q-learning algorithm to minimize the task scheduling time and optimize load balancing.

Zuo et al. [86] propose MOSACO, an algorithm based on ant colony optimization employed to minimize task completion times and costs using time-first and cost-first single-objective optimization strategies, respectively, and to maximize user quality of service and the profit of resource providers using an entropy optimization model.

Yang et al. [78] propose FOG-AMOSM, an adaptive neighborhood-based algorithm that aims at optimizing the total execution time and costs.

Abdel-Basset et al. [5] introduce a nature-inspired metaheuristic algorithm that integrates the marine predator's algorithm with the polynomial mutation mechanism (MHMPA) to minimize the makespan and carbon dioxide emission ratio.

Ali et al. [8] employ an optimization model based on a Discrete Non-dominated Sorting Genetic Algorithm II (DNSGA-II) to minimize both the makespan and total costs.

Hoseiny et al. [40] propose a priority-aware genetic algorithm, PGA, to optimize total computation time jointly, energy consumption, and the percentage of tasks completed before their deadline.

In this chapter, we propose three multi-objective task scheduling algorithms for the cloud-fog continuum, that minimize both the makespan and processing cost of the workflow, considering the QoS requirements of the applications. No previous work had ever addressed the multi-objective scheduling problem considering the diversity of the QoS requirements of applications running on the cloud-fog continuum. The FLAMSKE-INT algorithm is based on integer linear programming, FLAMSKE-RR implements an approximation to the exact solution given by FLAMSKE-INT, and FLAMSKE-RL is a reinforcement learning-based algorithm. Numerical results suggest that the FLAMSKE-RL overperforms both FLAMSKE-INT and FLAMSKE-RR in terms of optimality of scheduling and makespan, especially under network loads ranging from 60% to 95%, and yet keeping the execution times short.

6.2 Multi-objective task scheduling approaches based on classical optimization

This section describes FLAMSKE-INT and FLAMSKE-RR, two task schedulers based on integer linear programming designed to minimize the makespan and cost of a schedule. FLAMSKE-INT and FLAMSKE-RR are the multi-objective versions of the CASSIA-INT and CASSIA-RR algorithms presented in the previous chapter. Although the problem formulation provided below involves much of the information introduced in Section 5.3, we wanted to display the complete problem-solving analysis to help the reader follow this section.

The problem formulation considers that application tasks are executed one at a time. It is also assumed that when a task arrives at the processing host, the virtual machines (VMs), where tasks will run, are already activated.

The schedulers receive as input information about the availability of both processing elements on the fog and in the cloud and network links, the CoS of the application, and a DAG of the workflow specifying processing, RAM, storage, and communication demands of the application tasks. The following notation is used to represent an application composed of tasks and dependencies:

- m: number of tasks ($m \in \mathbb{N}$),
- I_j: processing demand of j^{th} task, expressed as the number of instructions to be processed ($I_j \in \mathbb{R}_+$),
- M_j: RAM demand of j^{th} task ($M_j \in \mathbb{R}_+$),
- S_j: storage demand of j^{th} task, expressed as the number of data units required by the task ($S_j \in \mathbb{R}_+$),
- $B_{i,j}$: number of data units transmitted between the i task and the j task ($B_{i,j} \in \mathbb{R}_+$),

The cloud-fog topology is represented by the following notation:

- n: number of hosts ($n \in \mathbb{N}$),
- o: task scheduler,
- R_k: available RAM capacity of the k^{th} host,
- D_k: available storage capacity of the k^{th} host,
- k_r: fastest processing node ($k_r \in V_H^{'}$),
- $\delta(k)$: set of processing elements linked to the k^{th} processing element in the network, including the processing element k,
- $TB_{k,h}$: time for transmitting a data unit on the link connecting the k^{th} processing element and the h^{th} processing element ($TB_{k,h} \in \mathbb{R}_+$),
- TS_k: time for transmitting a data unit on the link connecting the scheduler and the host k ($TS_k \in \mathbb{R}_+$),
- TI_k: the time the k^{th} host takes to execute a single instruction ($TI_k \in \mathbb{R}_+$),
- C_k: processing cost of the host k,
- B_{user}: user budget.

Moreover, three parameters are computed for each task. The first parameter, $P_{j,k}$, is the processing time of the task j on the host k, expressed as

$$P_{j,k} = I_j TI_k$$

The second parameter, C_{ij}, is the time taken for data transfer between two subsequent tasks i and j of the DAG, hosted, respectively, on the processing elements h and k.

$$C_{ij} = B_{ij} TB_{hk}$$

The third parameter, $L_{j,k}$, is the sum of the transmission delay, $d_{trans_{j,k}}$, and the propagation delay, $d_{prop_{j,k}}$. The transmission delay of all data of a task is computed by

$$d_{trans_{j,k}} = S_j TS_k$$

The propagation delay between the scheduler o and the host k, is given by the ratio between the distance between the two hosts divided by the propagation speed of the link, ω.

$$d_{prop_{o,k}} = d_{o,k}/\omega$$

Both transmission and propagation delays are considered from nodes on Fog layer 1 to nodes on Fog layer 1, Fog layer 2, on the cloud, or to processing elements in isolated hosts.

Moreover, time is discretized and defined as $T = \{1, ..., T_{max}\}$, where T_{max} is the time that the fastest host in $V_H^{'}$ would take to execute the tasks of the application serially. T_{max} is calculated by,

$$T_{max} = TS_{k_r} \sum_{j=0}^{m-1} S_j + m\left(d_{prop_{o,k_r}}\right) + TI_{k_r} \sum_{j=0}^{m-1} I_j + \max\left[B_{i,j} TB_{h,k}\right](m-1);$$

where,

$$TI_{k_r} = min\left(TI_{k|k \in V_H^{'}}\right)$$

The solution for the integer linear program is given by the value of the decision variable $x_{j,t,k}$ which is 1 if the j^{th} task should be processed at time t on the host k. The integer linear programming problem is formulated as follows:

Minimize

$$C_{max}$$

$$C_{proc} = \sum_{t \in T} \sum_{j \in V_D} \sum_{k \in V_H^{'}} (C_k P_{j,k}) x_{j,t,k}$$

Subject to

$$\sum_{t \in T} \sum_{k \in V_H^{'}} x_{j,t,k} = 1, \quad \forall j \in V_D; \tag{6.1}$$

$$\sum_{t \in T} (t + L_{j,k} + P_{j,k}) x_{j,t,k} \leqslant C_{max}, \quad \forall j \in V_D, k \in V_H^{'}; \tag{6.2}$$

$$\sum_{j \in V_D} \sum_{s=t}^{t+P_{j,k}-1} x_{j,s,k} \leqslant 1, \quad \forall k \in V_H^{'}, t \in T, \text{ where } t \leqslant T_{max} - P_{j,k}; \tag{6.3}$$

$$\sum_{s=1}^{t} x_{j,s,k} \leqslant \sum_{h \in \delta(k)} \sum_{s=1}^{t-L_{i,h}-P_{i,h}-C_{ij}} x_{i,s,h}, \quad \forall j \in V_D, k \in V_H^{'}, ij \in E_D, t \in T; \tag{6.4}$$

$$\sum_{t \in T} \sum_{j \in V_D} M_j x_{j,t,k} \leqslant R_k, \quad \forall k \in V_H^{'}; \tag{6.5}$$

$$\sum_{t \in T} \sum_{j \in V_D} S_j x_{j,t,k} \leqslant D_k, \quad \forall k \in V_H^{'}; \tag{6.6}$$

$$\begin{aligned} x_{j,t,k} \in \{0,1\}, \\ \forall j \in V_D, k \in V_H^{'}, t \in T; \end{aligned} \tag{6.7}$$

$$C_{proc} \leqslant B_{user}; \tag{6.8}$$

where C_{max} and C_{proc} represent two conflicting objectives, i.e. the makespan and the processing cost, respectively.

The objective is to find a schedule for V_D on $V_H^{'}$ that minimizes both C_{max} and C_{proc}. Constraint (6.1) establishes that each task (j) must be executed at once on a single host k. Constraint (6.2) ensures that the makespan is at least the longest completion time of the last operation of all tasks. Constraint (6.3) establishes that there is at most one task in execution on any host at any given time. Constraint (6.4) establishes that the execution of the j^{th} task cannot start until all its predecessor tasks have been completed and the data required by the j^{th} task have been transferred. Constraint (6.5) determines that the RAM occupied by the tasks processed on host k should not exceed its available RAM capacity. Constraint (6.6) establishes that the total amount of data occupied by tasks processed on host k should not exceed its available storage space capacity. Constraint (6.7) defines the domain for the variable $x_{j,t,k}$. Constraint (6.8) determines that the processing cost of the application must be lower than the user's budget.

The FLAMSKE-INT implements the integer linear program described above, while the FLAMSKE-RR applies the randomized rounding relaxation technique to reduce the execution time to produce a solution without significant loss of the quality of the schedule.

6.2.1 The FLAMSKE-INT scheduler

The FLAMSKE-INT scheduler is presented in Algorithm 6.1. The multi-objective integer linear program returns an exact schedule that gives the mapping of tasks on hosts. Integer programming is classified as an NP-complete problem. In addition, the time taken by the optimizer to solve the MOILP problem dominates the computational complexity of the FLAMSKE-INT scheduler.

6.2.2 The FLAMSKE-RR scheduler

The scheduling problem is an NP-Complete problem [65]. Techniques such as approximation algorithms provide solutions that guarantee solutions close to the optimum [14].

Algorithm 6.1: FLAMSKE-INT scheduler

Input : MOILP: Multi-objective integer linear program formulation.
Output: Schedule V_D on $V_H^{'}$ along with the schedule of the necessary hosts $k \in V_H^{'}$.

1 Solve the MOILP.
2 Let X be the solution schedule V_D on $V_H^{'}$, where $X = X_{j.k}$ and $X_{j.k} = \sum_{t \in T} x_{j,t,k}, \forall j \in V_D, k \in V_H^{'}$
3 Return the schedule.

The FLAMSKE-RR scheduler implements an approximation algorithm called *Randomized Rounding*, presented in Algorithm 6.2. First, the integer linear program is executed with the relaxation of the decision variables to turn the integer program into a linear program. The relaxation of the discrete time formulation consists of replacing the set {0,1} in constraint (7) by the interval [0,1]. The values found when solving this linear formulation represent the probability values of each task to be executed on a host. Algorithm 6.2 is executed P times, and the schedule with the shortest makespan is chosen. P must be defined in order to increase the chances of obtaining a good schedule. The execution time of the FLAMSKE-RR scheduler increases with the value of P. Algorithm 6.2 runs in $O(\alpha_{\mathscr{P}} + P \cdot (|V_D| \cdot |V_H^{'}| + |E_D| + |V_H^{'}|))$, where $\alpha_{\mathscr{P}}$ represents the time complexity to set each variable of the linear program, $X_{j,k}$, which is at least O($|V_D| \cdot |V_H^{'}|$).

Algorithm 6.2: FLAMSKE-RR scheduler

Input : MOILP: Multi-objective integer linear program formulation.
P: Number of drawings.
Output: Schedule of V_D in $V_H^{'}$ along with the schedule of the necessary hosts $k \in V_H^{'}$.

1 Execute the MOILP relaxed as a linear program (LP).
2 Let X be the solution schedule of V_D in $V_H^{'}$ for the LP, where $X = X_{j.k}$
3 **for** *P times* **do**
4 **for** *each task* $j \in V_D$ **do**
5 Let $X_{j.k}$ be the probability of mapping the task j on the host k, select a host at random where the task j should be executed, based on the previous mapping probability.
6 Add the task selected to a provisional list of tasks to be executed.
7 **if** *the provisional list meets the precedent constraint* **then**
8 Add the provisional list to the final list of schedules if it doesn't already exist
9 Keep the provisional list in the final list producing the schedule that jointly minimizes the makespan and the processing cost.
10 Return the schedule.

6.3 Multi-objective task scheduling approaches based on reinforcement learning

In Reinforcement learning (RL), an agent learns from its interactions with an environment to learn a policy that obtains the highest possible discounted cumulative reward intake.

6.3.1 Reinforcement Learning Agent

The RL agent uses the Q-learning technique to define where the application tasks should be processed according to the QoS requirements of the application. Q-learning is a model-free reinforcement learning method that updates the Q-value function after each iteration with the environment without estimating a model.

The cloud-fog continuum was modeled using a Markov Decision Process (MDP), consisting of the following elements:

- The state space S contains all possible states that the cloud-fog continuum can experience, where $s_t \in S$ is the state of the environment at time-step t. We define each state as a 3-tuple: s=(network load, class of service, number of network hosts). The cardinality of the state space is $|S = \{s_i\}| \equiv \{l \times c \times n\}$ where l is the number of loads taken into consideration for assessing the network utilization, c is the number of classes of service supported by the network, and n is the number of network hosts.

- The action space A, including all available actions that can be taken given a state s $\in$ S, where $a_t \in A$ is the action executed by the agent at time-step t. For each state $s_t \in S$, the RL-agent can take one action that conducts the transition to another state $s_j \in S$. The number of possible actions for a particular state s_i, is the number of processing layers, (P_{LR}), where the application can be scheduled according to Table 3.3. The cardinality of A is $|A| \equiv \sum_{s_i \in S} P_{LR}(s_i)$.

- A reward function $R(s_t, a_t)$ gives the expected immediate reward obtained by executing action a in state s. The reward for the agent is dependent on the makespan, the processing cost of the workflow, which are the objectives to be minimized in our problem, as well as the compliance with the QoS requirements of the application. For each action a_t that results in the next state s_t, we give a reward $R(s_t, a_t)$ according to the weighted fitness value function R defined as

$$R = [\omega/(m(S_{t+1}))] + [(1-\omega)/c(S_{t+1})] \tag{6.9}$$

  where ω determines the degree of contribution of each goal, and $m(S_{t+1})$ and $c(S_{t+1})$ represent the makespan and the processing cost after the current state S_t is transferred to the state S_{t+1}.

  If the processing layer, (P_{LS}), chosen by the action a_t, corresponds to a processing layer (P_{LR}) defined for the CoS of the application that is being scheduled, we give a positive reward R; otherwise, we give a penalty with the same R value.

- A strategy to balance exploitation and exploration to choose the actions. To obtain such balance, we use the ϵ-greedy strategy in which ϵ denotes the exploration rate. With $\epsilon = 1$, the agent always takes random actions for maximum exploration and, with $\epsilon = 0$, the agent always follows the currently known optimal policy with no random actions. Initially, we set the exploration rate to 1, so that the RL agent gets to explore the state space, and as it learns more about the environment, the ϵ is reduced so that the likelihood of exploration becomes less probable.

- A discount factor γ, with $0 \leq \gamma < 1$, gives more importance to immediate rewards compared to rewards obtained in the future.

- The Optimal policy of the MDP, with optimal being a maximized expected sum of rewards. The Q-value function depends on the selection of action in the state. Given the agent in state s_t and selecting action a_t, the Q-value function is expected to move to the best state and gain to maximize the total expected reward in the environment. The Q-value for the next state-action pair (s_{t+1}, a_{t+1}) at time t+1 can be calculated by:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[R_t + \gamma max_{a+1}Q(s_{t+1}, a_{t+1})] \tag{6.10}$$

  where α is the learning rate.

Figure 6.1 illustrates the join operation between the RL-agent and the cloud-fog continuum for multi-objective task scheduling. Initially, at time t, the environment is in state S_t. Then, the agent observes the current state and selects action A_t. After, the environment transitions to state S_{t+1} and grants the agent reward R_{t+1}. This process then starts over for the next time step, $t + 1$.

6.3.2 The FLAMSKE-RL scheduler

Due to the dynamism of the cloud-fog continuum, as well as the heterogeneity of both QoS requirements and the capacity of the processing devices, each state of the system requires multiple features to be represented. We split the RL solution to the multi-objective scheduling problem into two steps to obtain a low-dimensional representation of a state in the cloud-fog continuum,. The first, described in Algorithm 6.3, implements a Q-learning algorithm that finds the most suitable layer to process the application according to its QoS requirements. The second described in Algorithm 6.4, presents an exhaustive search algorithm that defines a scheduling plan that minimizes both the makespan and the processing cost of the application, using the hosts belonging to the layer chosen by the Q-learning algorithm in the first phase of FLAMSKE-RL.
The worst-case complexity of Algorithm 6.3 is derived next.

Lemma 1. *In Q-learning algorithms with a state space topology with a linear upper action bound* $b \in \aleph_0 \iff e \leq bn$ *for all* $n \in \aleph_0$*, where* e *is the cardinality of the action space and* n *the cardinality of the state space, the worst-case complexity is* $O(n^2)$

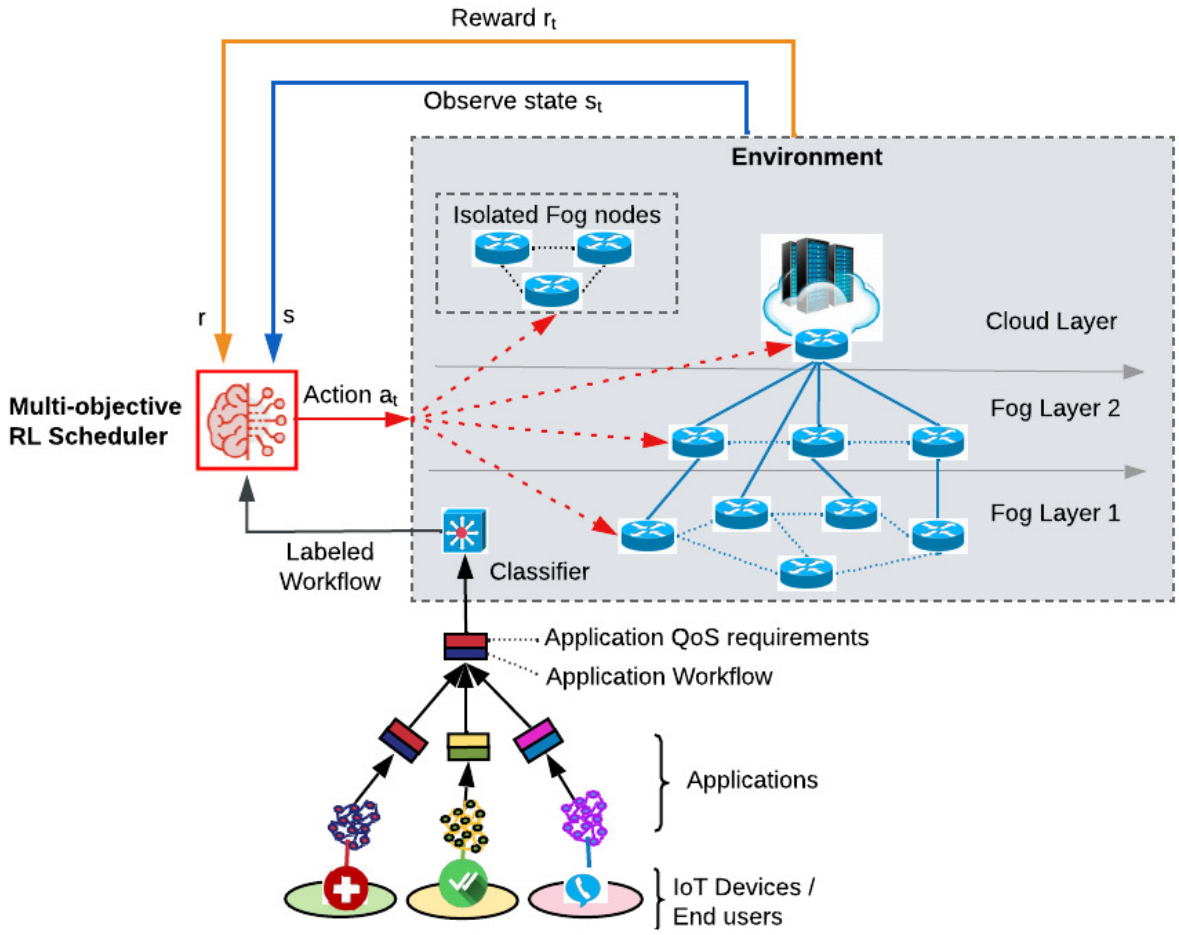


Figure 6.1: RL-based multi-objective task scheduling for the cloud-fog continuum.

Proof. see Page 103 in [48] ■

Theorem 1. *The worst-case complexity of Algorithm 6.3 is* $O(S^2)$

Proof. $P_{LR}(s_i) \leq S - 1$, for $S \in \aleph_0 \implies |A| \leq S^2$ ■

The worst-case complexity of Algorithm 6.4 is $O(m^2n^2)$.

6.4 Performance evaluation

This section assesses the performance of the FLAMSKE-INT, FLAMSKE-RR, and FLAMSKE-RL schedulers. Three metrics were evaluated, the makespan of the application when scheduled with one of the proposed schedulers, the processing cost of the schedule, and the execution time taken by the scheduler to give the schedule.

6.4.1 Simulation settings

The cloud-fog continuum was represented by a graph that describes the topology of the network. The topologies of the graphs are generated using the Barabási-Albert model with the same configuration parameters used in Table 5.4.

Algorithm 6.3: FLAMSKE-RL (Q-learning phase)

Input : Total number of learning episodes n, maximum number of steps per episode s, learning rate α, discount rate γ, exploration rate: ε, maximum exploration rate max_{ε}, minimum exploration rate min_{ε}, exploration decay rate λ, weighting factor between makespan and processing cost Ω, set of application workflow graphs *Apps*, set of topologies graphs *Topos*, all pair (app, topo) *SList*

Output: $\pi \approx \pi*$

```
for each (app,topo) ∈ SList do
    Initialize Q(s,a) = 0, ∀s ∈ 𝒮, ∀a ∈ 𝒜;
    for each episode do
        Initialize the starting state S_t;
        for each time step do
            Select A_t for S_t with policy derived from Q using ε-greedy exploration
             and exploitation method;
            Carry out action A;
            Observe reward R_{t+1} and next state S_{t+1};
            R_{t+1} ← R(S_t, A_t) ;
            Update Q_{t+1} by (6.10) ;
            S_t ← S_{t+1}
Return π(s) = arg max_a Q(s,a)
```

Algorithm 6.4: FLAMSKE-RL (Exhaustive search phase)

Input : DAG with a set of tasks V_D; Topology graph V_H; Q-value table resulting from FLAMSKE-RL in the Q-learning phase.

Output: Multi-objective schedule of V_D in $V_H^{'}$ along with the schedule of the necessary hosts $k \in V_H^{'}$.

```
Set the current state of the environment by using the current network load, the
 CoS of the application to be scheduled, and the number of available processing
 hosts
Select the processing layer where the application should be scheduled by choosing
 the action with the highest value for the current state based on the Q-Table.
for each task i ∈ V_D do
    for each task j ∈ V_D do
        if dependency between i and j exists then
            for each host k ∈ V_H' do
                for each host h ∈ V_H' do
                    if data transfer link between k and h exists then
                        Compute the finishing time of i^th task on host h considering
                         the time required to transfer data to the j^th task.
Consider the schedule if it produces the shortest execution time.
Compute the cost of the schedule that produces the shortest execution time.
Select the shortest and least expensive schedule.
Return the schedule.
```

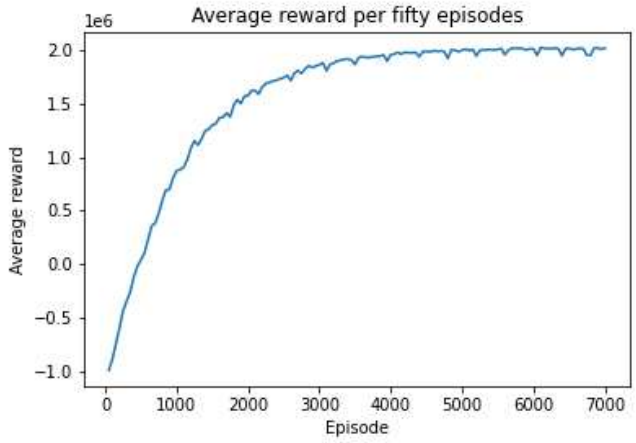


Figure 6.2: The convergence of the Q-learning phase of the FLAMSKE-RL scheduler.

We conduct experiments using workflow applications with 10 tasks for applications with seven different CoS. Characteristic values of each application task such as processing capacity, RAM memory, storage, and data to be transferred to the next task were assigned by employing a uniform probability distribution, within the intervals specified for each CoS in Table 6.2.

For each one of the seven CoS proposed in this thesis, different scenarios were generated by varying the number of hosts in the network and the network load. The number of hosts was 36, 72, 108, and 180, and the mean resource and bandwidth utilization were 0%, 20%, 40%, 60%, 80% and 95%. The utilization level of each node and link were randomly determined so that an average target utilization could be achieved.

Different values of P were tested for the FLAMSKE-RR; a value of 10,000 is recommended since no further improvement in results can be obtained by using a larger value. Also, we set the user budget to US$200.

Moreover, links with 2 Gbps, 1 Gbps, and 500 Mbps of bandwidth connect the scheduler with the Fog layer 1, the Fog layer 2, and the cloud, respectively. Two other 10 Gbps links connect the Fog layer 1 to the Fog layer 2, as well as connect the fog layer 2 to the cloud.

For the Q-learning step of the FLAMSKE-RL algorithm, we choose $\alpha = 0.001$, $\gamma = 0.99$, $max_\epsilon = 1$, $min_\epsilon = 0.01$, $\lambda = 0.001$, and $\omega = 0.5$. We trained the Q-learning algorithm during 7000 episodes until the required convergence was achieved, as evident from Figure 6.2.

The two linear programming-based schedulers were implemented in Java language using the IBM ILOG CPLEX Optimization Studio V12.10.0 to solve the ILP multi-objective model and its relaxation. The reinforcement learning scheduler was implemented by using Python 3.6. Algorithm 6.3 was trained on a GPU of Google Colaboratory. The other programs were executed on an AWS R5 instance Memory Optimized Intel Xeon 3.1 GHz EBS-Only with 20 GiB storage, 128 GB of RAM, and CentOS 7.0 operating system.

Table 6.3 details information about processing capacity, RAM memory, storage, and pricing of the processing hosts depending on which layer they are located.

Table 6.2: Task demands for each class of service.

CoS	Processing (MIPS)	RAM (GB)	Storage (GB)	File Transfer (GB)
Isolated Real-time Interactive	[100-350]	[0.1-0.5]	[1-3]	[0.1-0.3]
Conversational Streaming	[100-550]	[0.1-0.7]	[1-7]	[0.1-0.5]
CPU-Bound	[200-550]	[0.2-0.7]	[2-7]	[0.2-0.5]
Best-Effort	[100-550]	[0.1-0.7]	[1-7]	[0.1-0.5]

Table 6.3: Characteristics of the processing hosts distributed along the layers of the architectural model.

Device's Location	Processing Capacity (MIPS)	RAM (GB)	Storage (GB)	Processing Cost (USD/h)
Fog layer 1	1400	2.8	28	0.0510
Fog layer 2 / Isolated fog nodes	2800	4.2	42	0.1020
Cloud	3600	8.4	84	0.4080

6.4.2 Simulation results

In this subsection, we compare the makespan and the processing cost, generated by the three multi-objective task schedulers in each of the experiments explained in the previous subsection, as well as the execution time required by the schedulers to produce them. All figures presented in this subsection correspond to mean values with a 95% confidence interval.

Although we conducted experiments with seven different classes of service, we choose to show figures of the most representative class, i.e. "streaming" (CoS=5), to explain key results obtained with the rest of the classes. Results obtained with the rest of CoS are shown in Appendix A. Specific performance details observed for other classes are also further discussed throughout this section.

Figure 6.3 show the makespan for scenarios with 36, 72 and 108 nodes. In these figures, T_{max} corresponds to the time taken to execute all tasks in sequence. As the network load increases, the makespan of the application also increases.

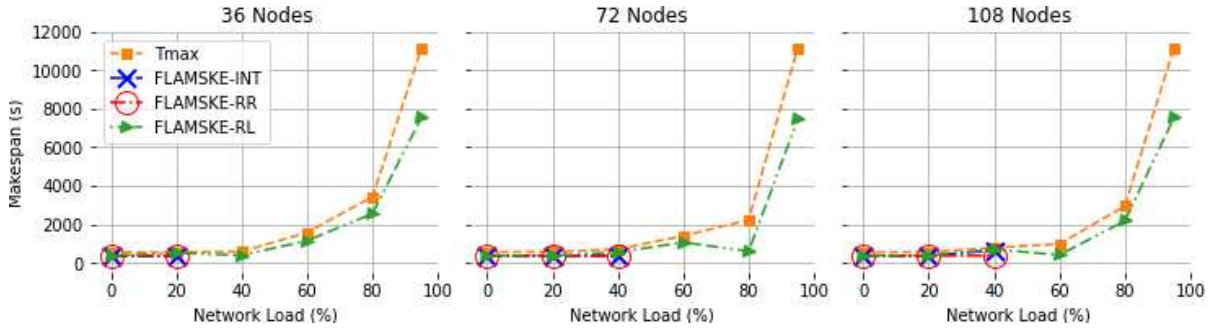


Figure 6.3: Makespan for CoS=5 with 36, 72 and 108 nodes.

Figure 6.4 show the processing cost for scenarios with 36, 72 and 108 nodes. In these figures, *Cost_ max* corresponds to the processing cost of executing all tasks in sequence.

The makespan and processing cost values generated by FLAMSKE-INT, FLAMSKE-RR, and FLAMSKE-RL are very close, and lower than their maximum reference values Tmax and Costmax.

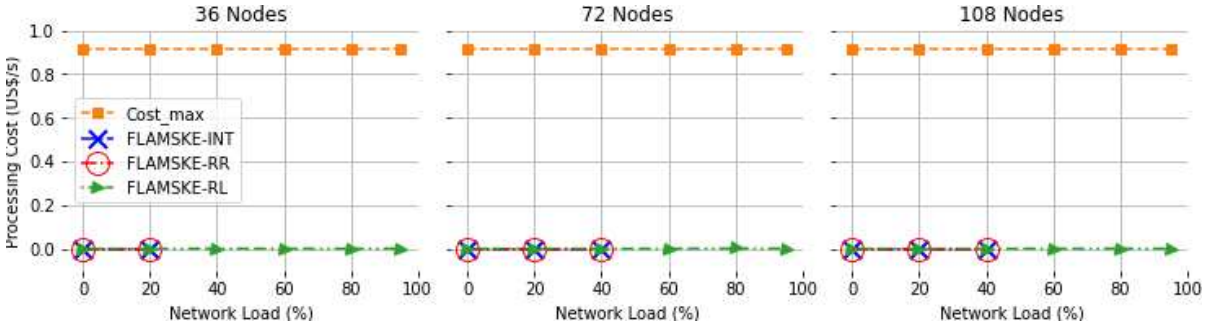


Figure 6.4: Processing cost for CoS=5 with 36, 72 and 108 nodes.

In the task scheduling of resource-hungry applications, such as the CPU-Bound class (CoS=6), FLAMSKE-INT and FLAMSKE-RR obtained solutions only for the scenario with 36 nodes running at no load. For the other classes, the largest network load for which FLAMSKE-INT and FLAMSKE-RR obtained optimal solutions was 60%. In contrast, FLAMSKE-RL obtained optimal solutions for all classes of service regardless of the number of nodes or the load percentage present in the network.

Figure 6.5 illustrates the execution time for scenarios with 36, 72 and 108 nodes. These figures show that when the number of network hosts increases, the FLAMSKE-INT scheduler needs more time to generate a schedule, while the time interval required by the FLAMSKE-RR and FLAMSKE-RL algorithms has minimal variation.

All the characteristics described above make the FLAMSKE-RL an ideal task scheduler to be implemented in cloud-fog environments dealing with a wide spectrum of applications, including those with latency constraints.

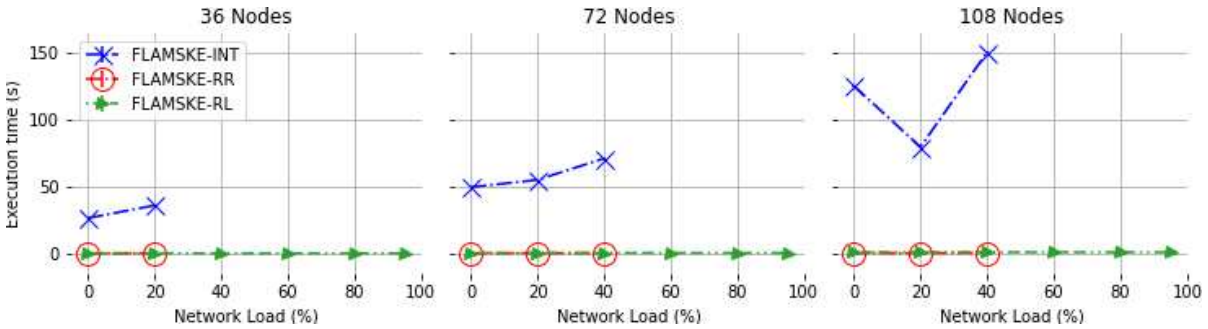


Figure 6.5: Execution time CoS=5 with 36, 72 and 108 nodes.

Chapter 7

Conclusion

This chapter highlights the main contributions and pointing out possible directions for conducting further research. Section 7.1 presents the summary of the thesis, revisits our contributions, and answers the proposed research questions. Then, Section 7.2 presents the future research directions.

7.1 Summary and contributions

Quality of service provisioning is the fundamental feature for the provisioning of applications in fog computing networks, since it enables administrators to prioritize traffic and resources to guarantee the compliance of the QoS requirements of each type of application, in particular those with delay constraints.

The first step towards ensuring QoS in fog computing is to understand the demands of applications arriving at the edge of the network. To this end, the traffic must be classified by efficient approaches that discriminate QoS requirements, allowing meaningful identification, and consequently, classification of the applications.

Another challenging process in fog computing is the processing of application tasks. In this operation, processing resources available along to the C2T continuum must be efficiently used, and scheduling play a fundamental role in this efficiency. Application scheduling should be able to decide on which resource each application task should run, given the demands of the applications and resource capacities, even in the presence of conflictant scheduling objectives, such as the minimization of both makespan and processing cost.

On other hand, although in the last decades artificial intelligence techniques have becoming increasingly versatile and powerful, to the point of permeating various areas of knowledge, their implementation to solve classification and optimization problems in cloud-fog environments, is still an emerging topic. One obstacle is the lack of real-world datasets.

In this thesis, we addressed the above challenges by studying, formulating, and developing QoS provisioning mechanisms that combine service differentiation and task scheduling, with mathematical optimization and artificial intelligence techniques to ensure compliance with the QoS requirements of the applications arriving at the fog network for

processing. To achieve this objective, we had to answer four research questions enumerated in Section 1.2, which are referred below for discussion.

- **Research question 1:** *How can a fog computing network provide different levels of service for selected applications according to their QoS requirements?*

  To answer this question, we first reviewed the literature in Chapter 3, pinpointed feasible applications to be empowered or developed with the emergence of fog computing. We listed the most relevant QoS requirements in the fog computing, namely bandwidth, delay sensitivity, packet loss, reliability, availability, security, data location, mobility support and, scalability. After that, we grouped applications running on fog into seven classes with distinct set of requirements. The classes defined were mission-critical, real-time, interactive, conversational, streaming, CPU-bound and, best-effort. In addition, we provide a mapping of the proposed classes of service and the layered architecture of the cloud-fog continuum.

  The proposed classes of service can be used to prioritize network traffic and processing demands by schedulers and resource allocation mechanisms. Prioritization will support the processing of delay sensitive applications, moving non real-time applications farther to the edge. By promoting load balancing among the layers of a fog, it is most likely that the fog will be able to support a higher number of requests, contributing to the scalability of the Fog. Moreover, the definition of Class of Service can facilitate the assignment of functionalities to different fog layers for the processing of typical demands of applications in each Class of Service. In addition, business models will largely benefit from the definition of Class of Services to fog computing.

- **Research question 2:** *How can the fog computing network identify the class of service an arriving application in search of processing belongs to?*

  This question was addressed in Chapter 4 by using Machine Learning (ML) classification algorithms as a tool for QoS-aware resource management in fog computing. We used the CoSs defined in Chapter 3. First, we built a synthetic database of fog applications from the definition of the intervals that each QoS requirement relevant for each class. Then, the dataset was pre-processed to convert prior useless data into new data that can be used by ML techniques. Next, a set of popular ML algorithms was selected and trained and tested, using the examples in the synthetic database to measure the degree of accuracy and efficiency in their prediction of the CoS an application belongs to. For this, the synthetic database was contaminated with three different levels of attribute noise. For each noise level, the classifier conducted training and testing to measure the degree of robustness.

  This ML-based classification methodology allows the implementation of CoS to manage the traffic in Fogs, which constitutes a first step in the definition of QoS provisioning mechanisms in the C2T continuum. Moreover, the integration of IoT, Fog Computing, and Cloud Computing requires efficient management strategies capable of facilitating resource management tasks such as scheduling, allocation, and federation. The classification of fog applications resolves all these issues; besides, classifying Fog Computing applications can facilitate the decision-making process for fog scheduler.

In addition, the proposed methodology can be easily modified for the classification of applications in networked systems.

- **Research question 3:** *How can the network scheduler decide the most suitable location for processing an application, whether on a the fog layer or in the cloud, to meet the QoS requirements of the application and minimize the makespan?*

  To decide the locality where application tasks should run, that is on the fog (e.g., for those tasks that belongs to mission-critical or delay sensitive applications), the cloud (e.g., for those tasks that require massive storage and heavy-duty computation), in Chapter 5, we introduced two scheduling algorithms based on an ILP formulation: an optimal one, CASSIA-INT, and an approximate one, CASSIA-RR. The schedulers differ from previous proposals by the consideration of the CoS of the application to make decisions about the most suitable location for processing an application. Both schedulers, in addition, seek a schedule with the smallest makespan.

  The effectiveness and efficiency of these schedulers were demonstrated in different scenarios, with different numbers of processing nodes, and utilization levels. The CASSIA-INT and CASSIA-RR algorithms outperform traditional scheduling algorithms such as Random and Round Robin in terms of makespan while not causing QoS violations. Results obtained showed that for all scenarios, the makespan values obtained from both schedulers were smaller than that of a serial execution. Furthermore, CASSIA-RR produces approximated results close to those given by CASSIA-INT in a much shorter time, and its use is recommended for cloud-fog systems in which fast decisions need to be made.

- **Research question 4:** *Compared with integer linear programming and approximative algorithms, can reinforcement learning be a promising technique for task scheduling in the cloud-fog continuum to meet the QoS requirements of the application and minimize both makespan and processing cost?*

  In Chapter 6, we studied the problem of multi-objective workflow scheduling in the cloud-fog continuum. We model the multi-objective task scheduling problem as a Markov Decision Process (MDP) theory, and made comparisons with different methods such as integer linear programming and approximative algorithms.

  Numerical results showed that the Reinforcement Learning (RL) algorithm overperforms those based on classical optimization in terms of optimality of scheduling and makespan, especially under extreme scenarios with network loads ranging from 60% to 95%, yet keeping short execution times.

  Unlike most other existing approaches, our multi-objective optimization models are capable of seeking for a trade-off between makespan and processing cost without prior experts' knowledge, while meeting the QoS requirements of the application, thus improving the performance of scheduling.

The management and operation of fogs present numerous challenges which call for innovative solutions. The interface and functionality assignment to fog layers should allow

efficient management and dynamic allocation of resources. Reliability schemes are necessary to assure that fogs will continuously provide low latency, connectivity, and processing even when failures occur. Moreover, self-adaptation and cognition in the management of fogs need to be understood towards the deployment of autonomic fogs. The QoS provisioning mechanisms introduced in this thesis can facilitate addressing all the challenges that lay ahead, since they help in coping with the requirements of the applications under limited availability of resources.

7.2 Future research directions

During the development of this research, ideas have emerged to advance the state-of-the-art in fog computing. Such ideas are listed below, and may guide new research projects.

Quality of service differentiation for fog computing: In this thesis, we proposed a set of classes of service for Fog Computing according to the QoS requirements of the potential fog applications. Future work should focus on the implementation of classes of service into existing devices of the fog computing architecture, to create self-adaptive and cognitive network mechanisms, which are fundamental for the deployment of autonomic fog systems in environments with limited availability of processing resources.

We also know that in the near term, improvements to existing fog applications and, of course, new fog applications will emerge. For this reason, we recommend to validate the intervals considered in the definition of each QoS requirement. It will probably be necessary to add new QoS requirements, classes, or use cases to our CoS specification to keep it up to date.

Machine learning-based QoS-aware classification of fog applications: To remove the burden of the analysis of the application requirements by the schedule, we introduced the use of ML classification algorithms as a tool for QoS-aware resource management in fog computing. We used as a basis the set of CoS defined in Chapter 3 for fog computing.

As future work, we envision the integration of an ML-based classification algorithm into the fog network scheduler, thus enabling the network scheduler to prioritize processing requests. It will also allow more delay sensitive demands to be satisfactorily fulfilled. We also suggest the inclusion of other feature selection methods such as Relief-F, CFS, MCFS, and the Student's t-test in the pre-processing stage for future classification studies in the context of fog computing. Moreover, we recommend a broad study on the distribution of noise for data analysis oriented to parameters related to cloud, fog and edge applications as well as network traffic.

Single-objective QoS-aware task scheduling in the cloud-fog continuum: In Chapter 5, we present two schedulers based on integer linear programming, that schedule tasks either in the cloud or on fog resources. The schedulers differ from existing ones by the use of class of services to select the processing elements on which the tasks should

be executed. Numerical results evince that the proposed schedulers outperform traditional ones without causing violation of QoS requirements. Further investigations, can be oriented to the integration of the CASSIA-INT and CASSIA-RR algorithms into the resource manager of a cloud-fog system. Further investigations can be oriented to integrate the CASSIA-INT and CASSIA-RR algorithms into the resource manager of a cloud-fog system. Also, it could be interesting to address the cross-layer application scheduling by orchestrating the placement of application tasks along the processing layers of the cloud-fog architectural model.

The proposed schedulers takes as input the information about the availability and the demand of resources in the network. However, this information depends on multiple sources of uncertainty, such as the medium used by IoT and end user devices to transmit data, fluctuating resource availability derived from the continuous changes in the network topology introduced by the mobility of IoT and end users, imprecise measurement tools, and the inability to accurately estimate the true demand of applications with real-time generated data.

Thus, vagueness in the estimation of available bandwidth can enlarge the makespan of an application when tasks are scheduled by a deterministic scheduler, which leads to increase in costs due to the need for leasing of processing power and additional power consumption costs.

Accordingly, to assess the fog network performance in more realistic conditions, an important question for future studies is to identify the main sources of uncertainty about communication demands in fog, and propose resource management mechanisms to deal with them.

Multi-objective QoS-aware task scheduling in the cloud-fog continuum: In Chapter 6, we demonstrated the potential of RL algorithms to efficiently obtain an optimal policy when the state space and action space are small. However, the multi-objective scheduling problem in the cloud-fog context is more complex due to the heterogeneity of both QoS requirements and the capacity of the processing devices, which usually require large space for making an optimal decision. Moreover, although the RL mechanism makes accurate scheduling decisions within the user budget, it takes a lot of time to reach an optimal decision as it explores and gains knowledge about the entire system state. In future work, we intend to implement a Deep Reinforcement Learning (DRL) methodology for improving the performance and learning speed of the entire multi-objective scheduling mechanism.

Another possible direction for future studies can be to include other optimization objectives such as load balancing, power consumption, latency, and bandwidth utilization in the multi-objective problem formulation.

Bibliography

[1] Aazam, M. and Huh, E. (2015a). Dynamic resource provisioning through Fog micro datacenter. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 105–110.

[2] Aazam, M. and Huh, E. N. (2015b). Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 687–694.

[3] Aazam, M., St-Hilaire, M., Lung, C., and Lambadaris, I. (2016). Mefore: Qoe based resource estimation at fog to enhance qos in iot. In *2016 23rd International Conference on Telecommunications (ICT)*, pages 1–5.

[4] Aazam, M., St-Hilaire, M., Lung, C. H., and Lambadaris, I. (2016). PRE-Fog: IoT trace based probabilistic resource estimation at Fog. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 12–17.

[5] Abdel-Basset, M., Moustafa, N., Mohamed, R., Elkomy, O. M., and Abouhawwash, M. (2021). Multi-Objective Task Scheduling Approach for Fog Computing. *IEEE Access*, 9:126988–127009.

[6] Agarwal, S., Yadav, S., and Yadav, A. (2016). An efficient architecture and algorithm for resource provisioning in fog computing. *International Journal of Information Engineering and Electronic Business*, 8:48–61.

[7] Alhamad, M., Dillon, T., and Chang, E. (2010). Conceptual sla framework for cloud computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies*, pages 606–610.

[8] Ali, I. M., Sallam, K. M., Moustafa, N., Chakraborty, R., Ryan, M. J., and Choo, K.-K. R. (2020). An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems. *IEEE Transactions on Cloud Computing*, pages 1–1.

[9] Ali, N. A., Taha, A. M., and Hassanein, H. S. (2013). Quality of service in 3gpp r12 lte-advanced. *IEEE Communications Magazine*, 51(8):103–109.

[10] Alsadie, D. (2021). A Metaheuristic Framework for Dynamic Virtual Machine Allocation With Optimized Task Scheduling in Cloud Data Centers. *IEEE Access*, 9:74218–74233.

[11] Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

[12] Batista, D. M., Fonseca, N. L. S. d., Granelli, F., and Kliazovich, D. (2007). Self-Adjusting Grid Networks. In *2007 IEEE International Conference on Communications*, pages 344–349.

[13] Bittencourt, L. F., Diaz-Montes, J., Buyya, R., Rana, O. F., and Parashar, M. (2017). Mobility-Aware Application Scheduling in Fog Computing. *IEEE Cloud Computing*, 4(2):26–35.

[14] Bittencourt, L. F., Madeira, E. R. M., and da Fonseca, N. L. S. (2015). Resource Management and Scheduling. In *Cloud Services, Networking, and Management*, pages 243–267. John Wiley & Sons, Inc. DOI: 10.1002/9781119042655.ch10.

[15] Bittencourt, L. F., Madeira, E. R. M., and Fonseca, N. L. S. D. (2012). Scheduling in hybrid clouds. *IEEE Communications Magazine*, 50(9):42–47.

[16] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, New York, NY, USA. ACM.

[17] Buttazzo, G. (2011). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series. Springer US, 3 edition.

[18] Byers, C. C. (2017). Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks. *IEEE Communications Magazine*, 55(8):14–20.

[19] Böhmer, M., Hecht, B., Schöning, J., Krüger, A., and Bauer, G. (2011). Falling Asleep with Angry Birds, Facebook and Kindle: A Large Scale Study on Mobile Application Usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 47–56, New York, NY, USA. ACM.

[20] Cai, D., Zhang, C., and He, X. (2010). Unsupervised Feature Selection for Multi-cluster Data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 333–342, New York, NY, USA. ACM.

[21] Cardellini, V., Grassi, V., Presti, F. L., and Nardelli, M. (2015). On qos-aware scheduling of data stream applications over fog computing infrastructures. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 271–276.

[22] Chekired, D. A., Khoukhi, L., and Mouftah, H. T. (2018). Industrial iot data scheduling based on hierarchical fog computing: A key for enabling smart factory. *IEEE Trans. Industrial Informatics*, 14(10):4590–4602.

[23] Chiang, M. and Zhang, T. (2016). Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, 3(6):854–864.

[24] Cisco Systems, Inc. (2020). Cisco annual Internet report (2018—2023). White Paper.

[25] Deng, R., Lu, R., Lai, C., and Luan, T. H. (2015). Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In *2015 IEEE International Conference on Communications (ICC)*, pages 3909–3914.

[26] Emeakaroha, V. C., Brandic, I., Maurer, M., and Dustdar, S. (2010). Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In *2010 International Conference on High Performance Computing & Simulation*, pages 48–54.

[27] Emeakaroha, V. C., Ferreto, T. C., Netto, M. A. S., Brandic, I., and De Rose, C. A. F. (2012). Casvid: Application level monitoring for sla violation detection in clouds. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 499–508.

[28] Fonseca, N. L. S. d. and Boutaba, R. (2015). *(Org.). Cloud Services, Networking, and Management*. 1. ed. Hoboken: John Wiley & Sons.

[29] Gao, Y., Zhang, S., and Zhou, J. (2019). A Hybrid Algorithm for Multi-Objective Scientific Workflow Scheduling in IaaS Cloud. *IEEE Access*, 7:125783–125795.

[30] García, S., Luengo, J., and Herrera, F. (2015). Dealing with Noisy Data. In *Data Preprocessing in Data Mining*, pages 107–145. Springer International Publishing, Cham.

[31] Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

[32] Guevara, J. C., Bittencourt, L. F., and da Fonseca, N. L. S. (2017a). Class of service in fog computing. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, pages 1–6.

[33] Guevara, J. C., Bittencourt, L. F., and Fonseca, N. L. S. d. (2017b). Class of service in fog computing. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, pages 1–6.

[34] Guevara, J. C., Torres, R. d. S., and Fonseca, N. L. S. d. (2020). On the classification of fog computing applications: A machine learning perspective. *Journal of Network and Computer Applications*, 159:102596.

[35] Gupta, H., Dastjerdi, A. V., Ghosh, S. K., and Buyya, R. (2016). iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. *arXiv:1606.02007 [cs]*. arXiv: 1606.02007.

[36] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1):389–422.

[37] Hao, Z., Novak, E., Yi, S., and Li, Q. (2017). Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing*, 21(2):44–53.

[38] He, X., Wang, K., Huang, H., Miyazaki, T., Wang, Y., and Sun, Y. (2018). Qoe-driven joint resource allocation for content delivery in fog computing environment. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6.

[39] Hobfeld, T., Schatz, R., Varela, M., and Timmerer, C. (2012). Challenges of qoe management for cloud applications. *IEEE Communications Magazine*, 50(4):28–36.

[40] Hoseiny, F., Azizi, S., Shojafar, M., Ahmadiazar, F., and Tafazolli, R. (2021). PGA: A Priority-aware Genetic Algorithm for Task Scheduling in Heterogeneous Fog-Cloud Computing. In *IEEE INFOCOM WKSHPS 2021*, pages 1–6.

[41] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*. arXiv: 1704.04861.

[42] Huber, P., Wiley, J., and InterScience, W. (1981). *Robust statistics*. Wiley New York.

[43] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360 [cs]*. arXiv: 1602.07360.

[44] Intharawijitr, K., Iida, K., and Koga, H. (2016). Analysis of fog model considering computing and communication latency in 5G cellular networks. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–4.

[45] Jain, R., Jain, R. K., and Jain (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, New York, edição: 1 edition.

[46] Jiahao, W., Zhiping, P., Delong, C., Qirui, L., and Jieguang, H. (2018). A Multi-object Optimization Cloud Workflow Scheduling Algorithm Based on Reinforcement Learning. In *Intelligent Computing Theories and Application*, pages 550–559. Springer, Cham.

[47] Khan, S., Parkinson, S., and Qin, Y. (2017). Fog computing security: a review of current applications and security solutions. *Journal of Cloud Computing*, 6(1):19.

[48] Koenig, S. and Simmons, R. G. (1993). Complexity analysis of real-time reinforcement learning. In *Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, pages 99–105, Washington, D.C. AAAI Press.

[49] Kohavi, R. (1995). A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[50] Kotb, Y., Al Ridhawi, I., Aloqaily, M., Baker, T., Jararweh, Y., and Tawfik, H. (2019). Cloud-Based Multi-Agent Cooperation for IoT Devices Using Workflow-Nets. *Journal of Grid Computing*.

[51] Kurose, J. F. and Ross, K. W. (2012). *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition.

[52] Liu, H. and Motoda, H. (2007). *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC.

[53] Liu, L., Fan, Q., and Buyya, R. (2018). A Deadline-Constrained Multi-Objective Task Scheduling Algorithm in Mobile Cloud Environments. *IEEE Access*, 6:52982–52996.

[54] Mahmud, R. and Buyya, R. (2016). Fog computing: A taxonomy, survey and future directions. *CoRR*, abs/1611.05539.

[55] Mahmud, R., Srirama, S. N., Ramamohanarao, K., and Buyya, R. (2019). Quality of experience (qoe)-aware placement of applications in fog computing environments. *Journal of Parallel and Distributed Computing*, 132:190 – 203.

[56] MATLAB (2018). Choose Classifier Options - MATLAB & Simulink. Available: https://www.mathworks.com/help/stats/choose-a-classifier.html [Accessed: 21/05/2018].

[57] McGregor, A., Hall, M., Lorier, P., and Brunskill, J. (2004). Flow Clustering Using Machine Learning Techniques. In Barakat, C. and Pratt, I., editors, *Passive and Active Network Measurement*, Lecture Notes in Computer Science, pages 205–214. Springer Berlin Heidelberg.

[58] Medina, A., Lakhina, A., Matta, I., and Byers, J. (2001). BRITE: Universal Topology Generation from a User's Perspective. Technical report, Boston University, Boston, MA, USA.

[59] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2018). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464.

[60] Naqa, I. E., Li, R., and Murphy, M. J., editors (2015). *Machine Learning in Radiation Oncology: Theory and Applications*. Springer International Publishing.

[61] OpenFog (2017). OpenFog Reference Architecture: OpenFog Consortium. Available: https://www.openfogconsortium.org/ra/ [Accessed: 24/05/2017].

[62] Oueis, J., Strinati, E. C., and Barbarossa, S. (2015). The Fog Balancing: Load Distribution for Small Cell Cloud Computing. In *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pages 1–6.

[63] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572.

[64] Pham, X.-Q. and Huh, E.-N. (2016). Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4.

[65] Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag, New York, 4 edition.

[66] Ren, Z., Lu, T., Wang, X., Guo, W., Liu, G., and Chang, S. (2020). Resource scheduling for delay-sensitive application in three-layer fog-to-cloud architecture. *Peer-to-Peer Networking and Applications*, 13(5):1474–1485.

[67] Roffo, G. (2016). Feature Selection Library (MATLAB Toolbox). *arXiv:1607.01327 [cs]*. arXiv: 1607.01327.

[68] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381 [cs]*. arXiv: 1801.04381.

[69] Shao, Y., Li, C., Fu, Z., Jia, L., and Luo, Y. (2019). Cost-effective replication management and scheduling in edge computing. *Journal of Network and Computer Applications*, 129:46 – 61.

[70] Skarlat, O., Nardelli, M., Schulte, S., and Dustdar, S. (2017). Towards qos-aware fog service placement. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 89–96.

[71] Souza, V. B. C., Ramírez, W., Masip-Bruin, X., Marín-Tordera, E., Ren, G., and Tashakor, G. (2016). Handling service allocation in combined Fog-cloud scenarios. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–5.

[72] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[73] Wang, K., Yin, H., Quan, W., and Min, G. (2018). Enabling Collaborative Edge Computing for Software Defined Vehicular Networks. *IEEE Network*, 32(5):112–117.

[74] Wang, S., Li, K., Mei, J., Xiao, G., and Li, K. (2017a). A Reliability-aware Task Scheduling Algorithm Based on Replication on Heterogeneous Computing Systems. *Journal of Grid Computing*, 15(1):23–39.

[75] Wang, S., Urgaonkar, R., He, T., Chan, K., Zafer, M., and Leung, K. K. (2017b). Dynamic service placement for mobile micro-clouds with predicted future costs. *IEEE Trans. Parallel Distrib. Syst.*, 28(4):1002–1016.

[76] Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., and Xie, H. (2019). Multi-Objective Workflow Scheduling With Deep-Q-Network-Based Multi-Agent Reinforcement Learning. *IEEE Access*, 7:39974–39982.

[77] Wu, L., Garg, S. K., Buyya, R., Chen, C., and Versteeg, S. (2013). Automated sla negotiation framework for cloud computing. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 235–244.

[78] Yang, M., Ma, H., Wei, S., Zeng, Y., Chen, Y., and Hu, Y. (2020). A Multi-Objective Task Scheduling Method for Fog Computing in Cyber-Physical-Social Services. *IEEE Access*, 8:65085–65095.

[79] Yang, S. (2017). Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, 55(8):21–27.

[80] Zeng, D., Gu, L., Guo, S., Cheng, Z., and Yu, S. (2016). Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System. *IEEE Transactions on Computers*, PP(99):1–1.

[81] Zhang, G., Shen, F., Yang, Y., Qian, H., and Yao, W. (2018). Fair Task Offloading among Fog Nodes in Fog Computing Networks. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6.

[82] Zheng, W., Yan, W., Bugingo, E., and Zhang, D. (2018). Online Scheduling to Maximize Resource Utilization of Deadline-Constrained Workflows on the Cloud. In *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, pages 98–103.

[83] Zhong, S., Khoshgoftaar, T. M., and Seliya, N. (2004). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems*, 19(2):20–27.

[84] Zhu, X. and Wu, X. (2004). Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review*, 22(3):177–210.

[85] Zhu, Z., Zhang, G., Li, M., and Liu, X. (2016). Evolutionary Multi-Objective Workflow Scheduling in Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1344–1357.

[86] Zuo, L., Shu, L., Dong, S., Chen, Y., and Yan, L. (2017). A Multi-Objective Hybrid Cloud Resource Scheduling Method Based on Deadline and Cost Constraints. *IEEE Access*, 5:22067–22080.

Appendix A

Simulation results of multi-objective QoS-aware task scheduling in the cloud-fog continuum

This appendix presents the results obtained from experiments related to multi-objective task scheduling in the cloud-fog continuum, described in Chapter 6. Figures A.1-A.7, illustrate makespan, processing cost, and execution time results obtained for the classes of service proposed in this thesis, i.e. mission-critical (CoS=1), real-time (CoS=2), interactive (CoS=3), conversational (CoS=4), streaming (CoS=5), CPU-Bound (CoS=6), and best-effort (CoS=7), respectively.

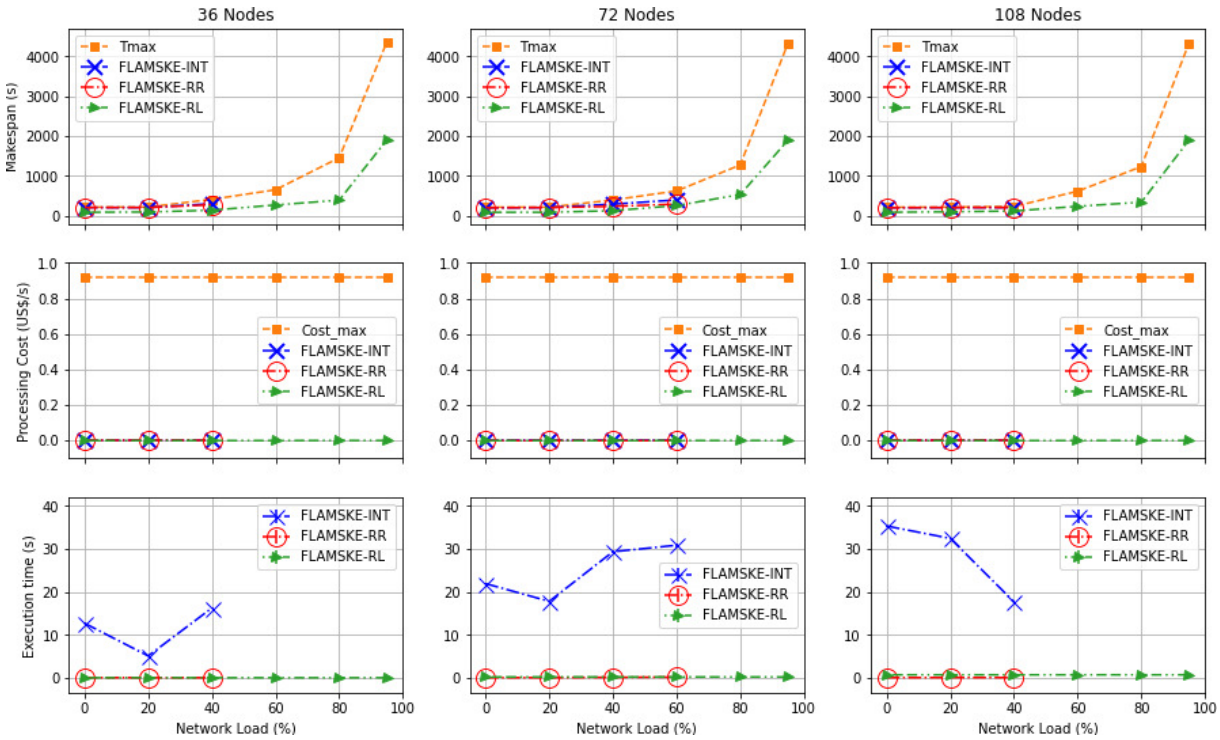


Figure A.1: Makespan, processing cost and execution time for CoS=1 with 36, 72 and 108 nodes.

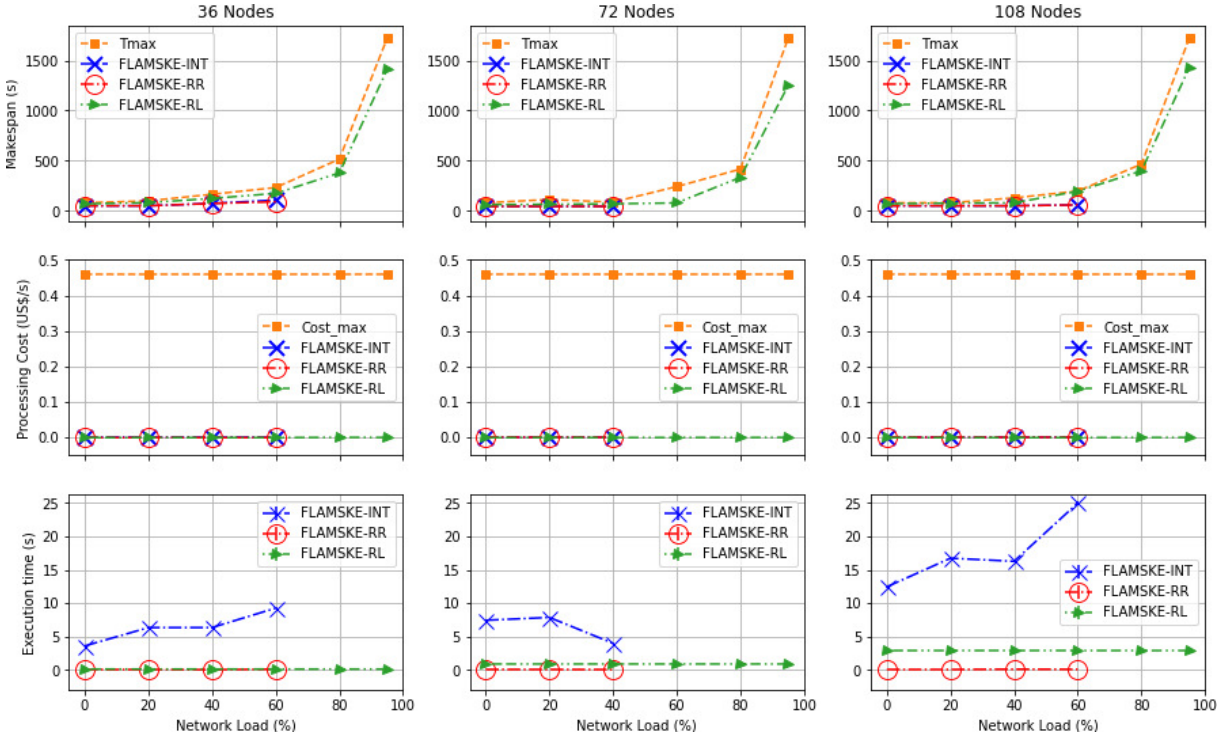


Figure A.2: Makespan, processing cost and execution time for CoS=2 with 36, 72 and 108 nodes.

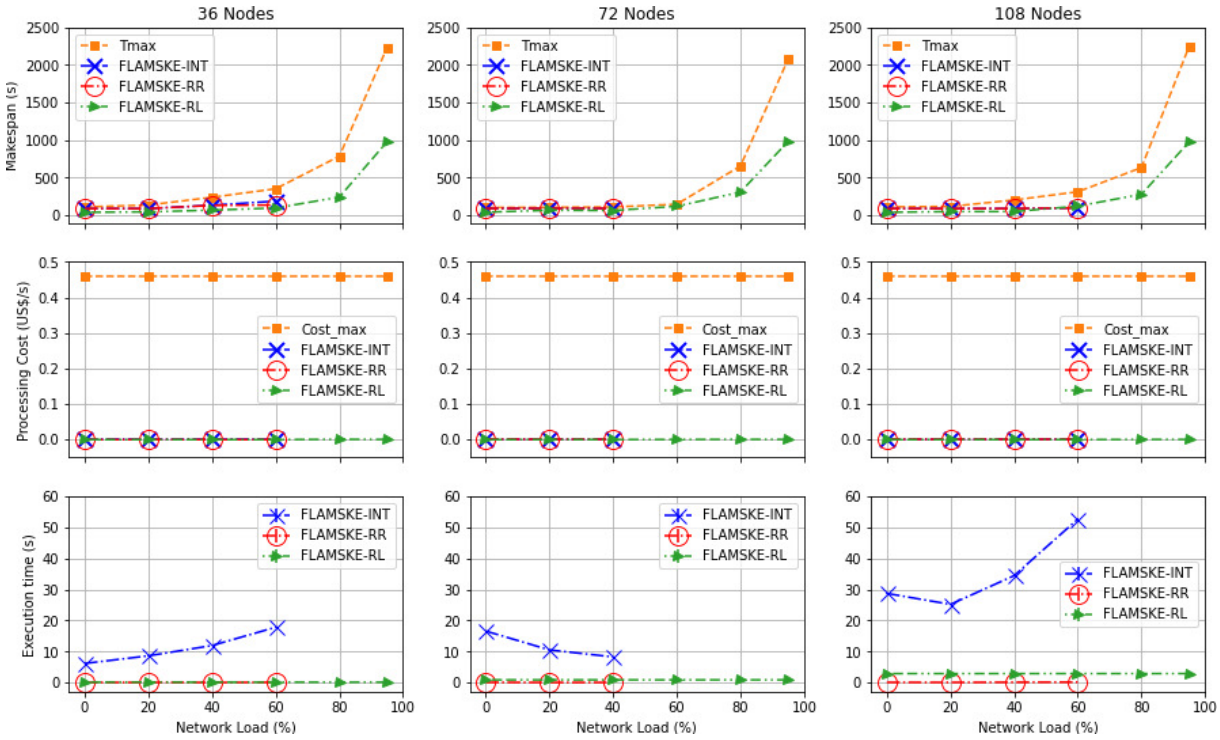


Figure A.3: Makespan, processing cost and execution time for CoS=3 with 36, 72 and 108 nodes.

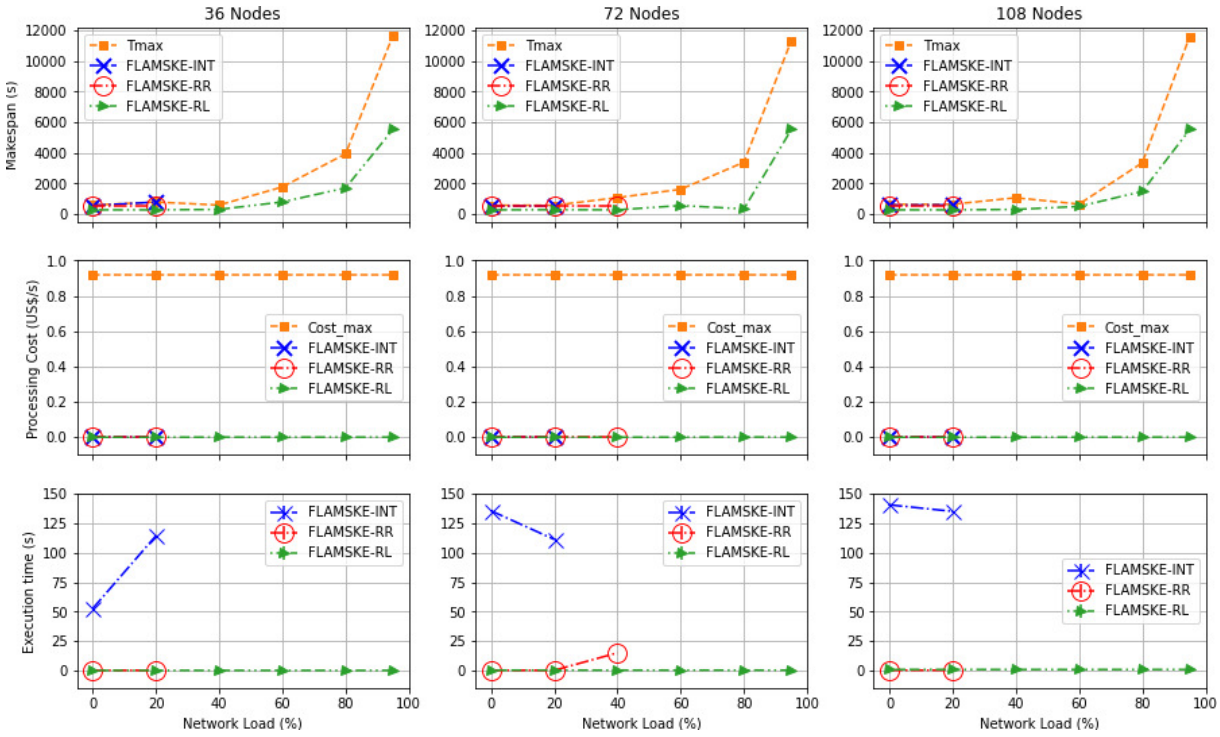


Figure A.4: Makespan, processing cost and execution time for CoS=4 with 36, 72 and 108 nodes.

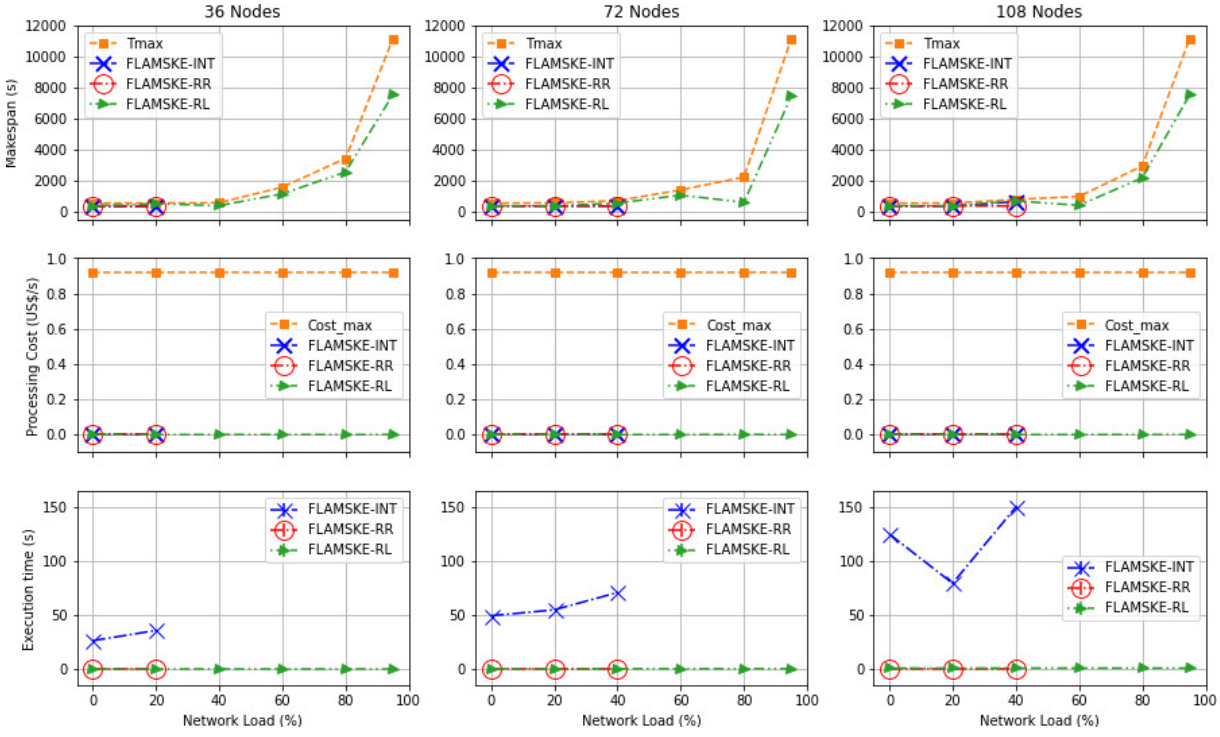


Figure A.5: Makespan, processing cost and execution time for CoS=5 with 36, 72 and 108 nodes..

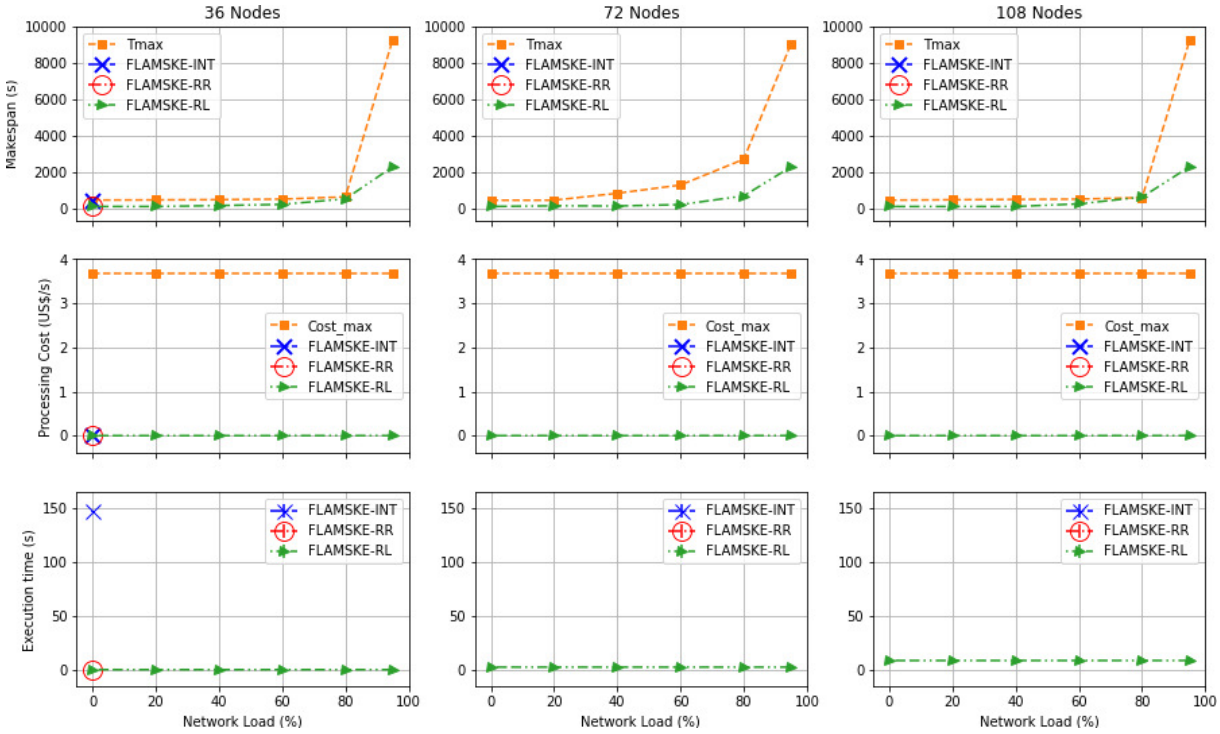


Figure A.6: Makespan, processing cost and execution time for CoS=6 with 36, 72 and 108 nodes.

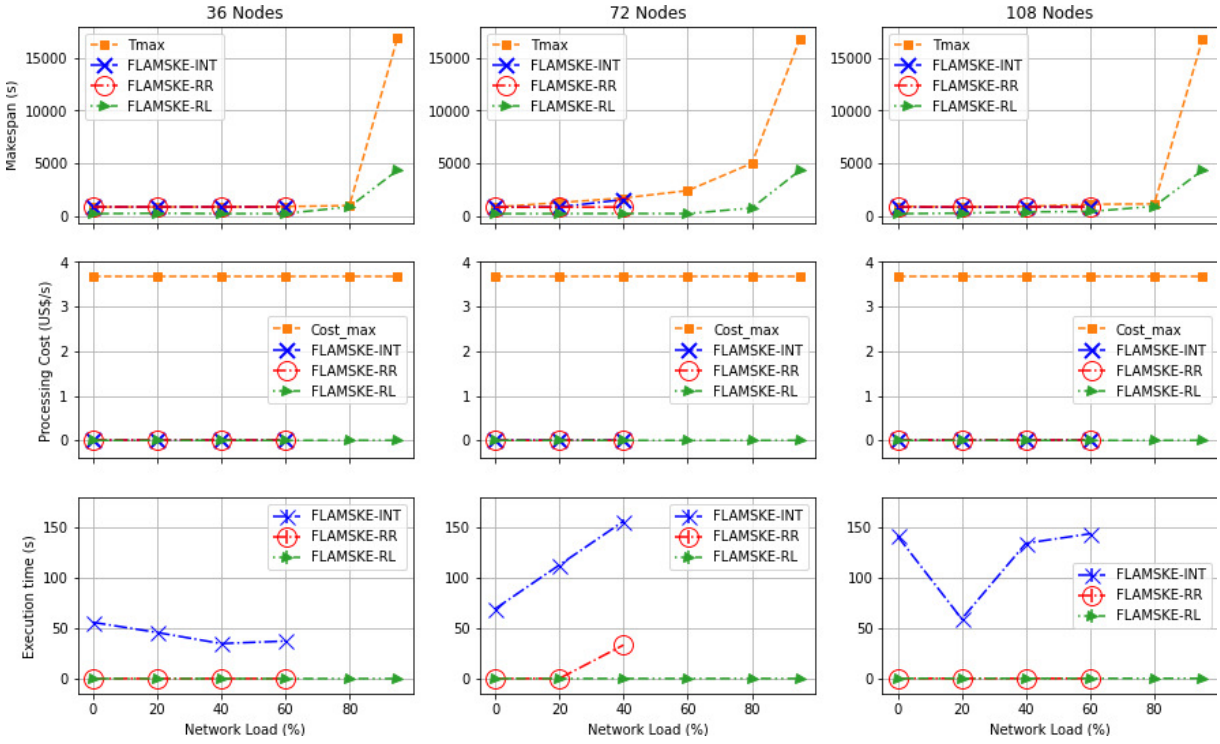


Figure A.7: Makespan, processing cost and execution time for CoS=7 with 36, 72 and 108 nodes.