UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Resource Allocation Strategies in Highly Distributed and Heterogeneous Computing Systems

Francescomaria Faticanti

Advisor

Prof. Francesco De Pellegrini

University of Avignon

Co-Advisor

Dr. Domenico Siracusa

Fondazione Bruno Kessler

November 2021

Abstract

The spread of IoT devices has led to the design of new extensions of cloud computing, such as fog computing, providing IoT applications with reduced latency, location-awareness and mobility support. The term cloud-to-things continuum in this context refers to the fact that the computation is no longer confined to a few data centers but workloads can be displaced from the central cloud to the edge of network involving multiple infrastructure owners and several devices with different computational characteristics. This heterogeneity impacts the capability of the infrastructure owner of satisfying the QoS requirements of clients. Consequently, solving the placement and orchestration problems among the cloud-to-things continuum becomes key to ensure the profitability for the involved stakeholders. This thesis focuses on the algorithmic solutions for the problem of placement and the orchestration of microservice-based applications in such a distributed and heterogeneous context.

On one hand, the placement problem involves the design of efficient solutions for the deployment of applications on a fog infrastructure, a type of problem which typically is NP-hard, even assuming a complete knowledge of applications' requirements and resource availability. In this thesis, the focus is on the design of approximated solutions for the NP-hard problems behind such resource allocation tasks.

The orchestration of fog applications, on the other hand, deals with the maintenance of applications' QoS requirements under partial information about applications requests arrivals. For each applications' module, orchestration algorithms involve the decision of deployment either in fog or in cloud as the applications requests vary over time. In order to deal with this problem, we developed solutions based on stochastic optimisation techniques.

The proposed methods outperform standard cloud-native solutions and suggest new approaches for inter-operability between different fog regions. Additionally, numerical results confirm the scalability properties of all the proposed solutions and their efficiency in terms of infrastructure owner's costs, for the placement side, and in terms applications' QoS, for the orchestration part.

To Agata, Fausto, Marisa, and Ugo

Acknowledgements

With these few lines I would like to thank all the people who have followed me along this journey.

First of all, I wish to thank my advisor, Francesco, for being a mentor, a colleague and a friend, for the endless passion that he transmits in his work. I promise to treasure all the precious teachings I received from him during these years. Firstly, I will start to avoid taking cable cars to reach the top of mountains.

Thanks to Domenico for his availability, kindness and for being always able to wear others' shoes, a great quality to lead a research group. He always proposed me new exciting challenges and opportunities that let me grow professionally. I promise to exploit this experience in the future.

My gratitude goes to my Ph.D. referees, Prof. Rosa Figuereido and Prof. Daniel Sadoc Menasche, for their insightful comments that helped to improve the quality of this thesis.

I would like to thank all my co-authors. They are the evidence that my research is primarily made by people sharing their knowledge. This is a fundamental reason for which I chose this path.

Thanks to all the present and past RiSING members for their support and their limitless knowledge that they patiently shared with me.

There are no words to express my gratitude towards my family, just feelings. Thanks to my parents for their endless love, for always believing in me and for always being there for me. Thanks to my sister Chiara, the best part of me. Thanks for being who you are.

Last but not least, I would like to thank all my lifelong friends and the ones I got during these three years. Thank you for for always listening to me and showing me your support and comprehension.

Activity Report

Research/Study Activities

The activities conducted during the doctoral programme are the following:

- Three years of research in several topics related to fog computing and distributed systems in the RiSING group at Fondazione Bruno Kessler.
- Participation in the research activities of the EU H2020 DECENTER project (grant agreement n. 815141).
- Visiting period (26/05/2019 - 26/06/2019) at Laboratoire Informatique d'Avignon (LIA), University of Avignon (France) to work on optimal control of fog applications' orchestration.
- Participation in the local arrangement of WiOpt 2019 (03/06/2019 - 07/06/2019).
- Session chair at EdgeCom 2019, Paris (21/06/2019 - 23/06/2019).
- Internship period (02/02/2020 - 02/05/2020) in Nokia Bell Labs, Paris-Saclay (France) under the supervision of Lorenzo Maggi, to work on stochastic optimization methods for the joint caching and orchestration of fog applications.
- Participation in the writing of project proposals, both EU and industrial.
- Teaching activities for the following classes at University of Trento (Italy):
  - "Wireless Networks", academic year 2017/2018, second semester. Two-hours lecture about "Fog Computing";
  - "Cloud and Fog Computing", academic years 2018/2019, 2019/2020, 2020/2021, second semester. Two-hours lectures about "Resource allocation in Cloud and Fog Computing".
- Co-supervision of master students from the Department of Information Engineering and Computer Science (DISI) at University of Trento for their master theses. In particular:

  - Valentino Armani, "A Cost-Effective Workload Allocation Strategy for Cloud-Native Edge Services", March 2021.
  - Riccardo Capraro, "Service-aware autoscaling of cloud-native workloads", March 2021.
  - Marian Alexandru Diaconu, "Serverless Caching Systems: Adapting a Large Objects Serverless Distributed Cache to manage Small Objects", March 2021.
  - Matteo Caliandro, "Workload Sceduling on Autonomous Agents", in progress.
- Reviewing activities for several international journals and conferences.

List of Publications

During the doctoral programme the following peer-reviewed conference and journal papers were published:

1. F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro and S. Cretti "*Cutting Throughput with the Edge: App-Aware Placement in Fog Computing*", 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom) 2019 Jun 21 (pp. 196-203), IEEE.

2. F. Pederzolli, F. Faticanti and D. Siracusa, "*Optimal design of practical quantum key distribution backbones for securing coretransport networks*", 2020, Quantum Reports, 2(1), pp.114-125.

3. F. Faticanti, M. Savi, F. De Pellegrini, P. Kochovski, V. Stankovski and D. Siracusa, "*Deployment of Application Microservices in Multi-Domain Federated Fog Environments*, 2020 International Conference on Omni-layer Intelligent Systems (COINS) (pp. 1-6), IEEE.

4. F. De Pellegrini, F. Faticanti, M. Datar, E. Altman and D. Siracusa, "*Optimal Blind and Adaptive Fog Orchestration under Local Processor Sharing*", 2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT) (pp. 1-8), IEEE.

5. F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro and S. Cretti, "*Throughput-Aware Partitioning and Placement of Applications in Fog Computing*", 2020, IEEE Transactions on Network and Service Management, 17(4), pp.2436-2450.

6. F. Faticanti, J. Zormpas, S. Drozdov, K. Rausch, O. García, F. Sardis, S. Cretti, M. Amiribesheli and D. Siracusa, "*Distributed Cloud Intelligence: Implementing an ETSI MANO-Compliant Predictive Cloud Bursting Solution Using Openstack and Kubernetes*", 2020, September, International Conference on the Economics of Grids, Clouds, Systems, and Services (pp. 80-85). Springer, Cham.

7. D. Tovazzi, F. Faticanti, D. Siracusa, C. Peroni, S. Cretti and T. Gazzini, "*GEM-Analytics: Cloud-to-Edge AI-Powered Energy Management*", 2020, September, International Conference on the Economics of Grids, Clouds, Systems, and Services (pp. 80-85). Springer, Cham.

8. P. Kochovski, V. Stankovski, S. Gec, F. Faticanti, M. Savi, D. Siracusa and S. Kum, "*Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing*", 2020, Journal of Grid Computing, pp.1-18.

9. F. Faticanti, D. Santoro, S. Cretti and D. Siracusa, "*An Application of Kubernetes Cluster Federation in Fog Computing*", to appear in Proceedings of the 24th Conference on Innovation in Clouds, Internet and Networks (ICIN 2021), BEST DEMO PAPER AWARD.

10. F. Faticanti, L. Maggi, F. De Pellegrini, D. Santoro, and D. Siracusa, "*Fog Orchestration meets Proactive Caching*", to appear in Proceedings of IFIP/IEEE International Symposium on Integrated Network Management 2021 (IM 2021), AnNet 2021 (Workshop).

11. L. A. D. Knob, F. Faticanti, T. C. Ferreto, and D. Siracusa, "*Community-based placement of registries to speed up application deployment on Edge Computing*", to appear in Proceedings of the 9th IEEE International Conference on Cloud Engineering IC2E 2021.

12. F. Faticanti, F. Bronzino, and F. De Pellegrini, "*The case for admission control of mobile cameras into the live video analytics pipeline*", to appear in Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges 2021.

13. F. Faticanti, M. Savi, F. De Pellegrini, and D. Siracusa, "*Locality-aware Deployment of Application Microservices for Multi-Domain Fog Computing*, under review.

14. V. Armani, F. Faticanti, S. Cretti, S. Kum, and D. Siracusa, "*A Cost-Effective Workload Allocation Strategy for Cloud-Native Edge Services*, under review.

Material from papers 1,3,4,5 and 10 is described in this thesis. Ongoing works include the extension of works presented in papers 4,5 and 10.

List of Collaborations

I had the precious opportunity, during the doctoral programme, to start collaborations with the following researchers all around the world:

- Lorenzo Maggi, Nokia Bell Labs, Paris, France;

- Francesco Bronzino, Université Savoie Mont Blanc, France;
- Eitan Altman, INRIA, France;
- Mandar Datar, INRIA, France;
- Marco Savi, University of Milano-Bicocca, Italy;
- Luis Augusto Dias Knob, PUCRS, Brazil;
- Tiago Coelho Ferreto, PUCRS, Brazil;
- Petar Kochovski, University of Ljubljana, Slovenia;
- Vlado Stankovski, University of Ljubljana, Slovenia;
- Ivan Pashchenko, University of Trento, Italy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The diffusion of remote sensors, mobile and IoT devices have inspired the design of new extensions of the cloud computing paradigm. The main problem is related to the huge amount of data generated by these devices resulting in bottlenecks for network communications and long response times due to high latencies between the devices and the cloud. IoT applications, especially real-time applications, present strict QoS requirements in terms of time responsiveness and bandwidth consumption [83]. Cloud computing concentrates all the computation in a few data centers resulting to be not adequate to handle the huge amount of data and requests generated by that applications.

In order to mitigate this kind of problems for IoT applications, new paradigms, such as fog computing [23], have been proposed. The idea behind fog computing is to add a new computation level between the cloud and the things in order to reduce costs, waiting times, and the demand generated towards the cloud. In this manner, a continuum of networking and computational resources is created between the things and the cloud. In order to exploit the benefits of such a distributed paradigm, applications are not deployed entirely in cloud or in fog. Rather, they can be distributed along the Cloud-to-thing continuum. For example, depending on specific requirements, part of a given application can be deployed close to the data source, and part in the cloud. This suggests the adoption of a different architectural design with respect to the monolithic one for applications in this new context.

Nowadays, the main trends in the design and development of cloud-native applications are microservice-oriented service architectures [84]. Microservice-oriented applications consist of a cascade of loosely-coupled components/modules (that is, the microservices) that can be containerized independently. Each microservice performs specific computations on input data and forwards the resulting output to other microservices downstream for further processing. Microservice design is preferred to monolithic application development because applications adopting such an architecture deliver same intended

functionalities but attain higher degree of flexibility, reliability and scalability [56]. In fog computing, on the other hand, available resources are geographically spread so that a microservice-oriented architecture appears a natural choice for fog-native applications, since it is possible to split them into components to be executed along the Cloud-to-things continuum.

It is worth noting that all the current proposed solutions, such as [2, 3], offer support only for the computational part, whilst the networking part is managed by telecommunication operators. This suggests that the distribution of applications on such a multiple-owners scenario is a fundamental problem both for the applications' users, who can have high QoS requirements, and the infrastructure owners whose objective is to minimize the deployment costs. As we will show in next chapters, the problems of how and where to deploy each module of each application on a distributed fog infrastructure can be seen as resource allocation problems, and they play a fundamental role in finding a good balance between the guarantee of all the applications' requirements, and the costs of management of such distributed infrastructures.

1.2 Contributions and Structure of the Thesis

The main objective of this thesis is to **contribute to the study and the design of new resource allocation algorithms in distributed and heterogeneous computing systems such as fog computing**. In this thesis we highlight and tackle the main problems arising in this context concerning the applications' deployment and the QoS maintenance. More in detail, this thesis work focuses on two main aspects: i) the static deployment, namely the *placement* of microservice applications in a region-based fog infrastructure via resource allocation algorithms that approximate well-known NP-hard problems, and ii) the dynamic deployment, namely *orchestration*, of applications involving the use of stochastic optimization techniques to deal with the high variation of demands for the applications deployed on the system. The first aspect will optimize the revenue of the infrastructure owner, while the latter guarantees strict delay constraints ensuring a good level of QoS for the applications' users.

In particular, the next chapter introduces the fog computing paradigm and the related challenges in terms of resource allocation needs for the placement and the orchestration of fog native applications.

Table 1.1 resumes the main aspects considered in the remaining chapters. In Chapter 3 we introduce the problem of applications' placement in a region-based and single-domain fog infrastructure with limited computational and networking resources. Applications are represented as Directed Acyclic Graphs (DAGs), and the main objective is to maximize the network operator's revenue maximizing the number of applications deployed on the infrastructure. The main problem results to be NP-hard and a heuristic algorithm is proposed with relevant numerical results.

Table 1.1: Table of chapters

		Chapter 3	Chapter 4	Chapter 5	Chapter 6
Problem	*Placement*	✓	✓		
	Orchestration			✓	✓
Objective	*Cost Min*		✓		✓
	Revenue Max	✓			
	Delay Min			✓	
App Structure	*Single Container*				✓
	Pipeline			✓	
	DAG	✓	✓		
Fog Domain	*Single Domain*	✓		✓	✓
	Multiple Domains		✓		

Chapter 4 presents the problem of microservice-based applications deployment in an enhanced scenario with an infrastructure divided in different domains. The main objective is to minimize the deployment cost of the infrastructure owner trying to satisfy all the applications' constraints. In particular, these constraints involve locality requirements, *e.g.,* a particular request for a device located in a different domain. Given the NP-hardness of the problem we provide heuristic approaches that outperform baseline solutions.

Chapter 5 tackles the problem of application orchestration in a dynamic environment where we have a set of pipeline applications deployed on the fog infrastructure and the objective is to minimize the total delay experienced by all the applications under budget constraints for cloud deployments. The total delay is given by the sum of the local processing time and the fixed processing time in cloud. In this chapter, we provide a model for the processing delay on a local server that shares its computational capacity among all the applications' modules deployed on it. Furthermore, we provide a well defined structure of problem's solution designing an optimal algorithm that works in polynomial time. Finally, we present a stochastic approximation approach for the case noisy arrival rates of the applications.

In Chapter 6 we present a new proactive caching framework to foster the orchestration of applications in fog computing. The aim is to minimize the deployment cost in cloud while satisfying the applications' requirements. We show that proactively caching applications' containers on fog servers is effective to match the expected activation pattern while optimizing load balancing via container replication. To this aim we design a one-step look-ahead policy to minimize the total running cost taking in input the prediction of future arrival rates for each application. Experimental results demonstrate the potential of this new approach over traditional caching and placement baselines.

Chapter 2

Fog Computing

This chapter presents the Fog Computing context, including technological novelties introduced by this new paradigm, and all the related challenges concerning the orchestration and the placement of applications for the users' QoS maintenance and network operators' revenue.

2.1 From a centralized Cloud to a decentralized and distributed Fog

Fog computing is a new technology that extends the cloud in order to tackle the problem of data explosion in the IoT domain [23]. The main idea is to move the computation close to the edge of the network, thus reducing the problem of bottlenecks in the network infrastructure in data retrieval from network devices, and mostly, reducing deployment costs in cloud. In this manner, the amount of data sent to the cloud is reduced, resulting in a reduced bandwidth consumption and, hence, in reduced response times. Furthermore, proximity to data sources lowers the round-trip time between objects and the backend of processing applications. Further motivations for the adoption of fog computing are the standardization of IoT services and privacy issues in data usage, which may be confined to specific geographical areas [23, 112].

Figure 2.1 captures the differences with respect to the cloud computing paradigm. A typical fog infrastructure can consist of a layered architecture, including a central cloud, a series of edge units (fog nodes), gateways to connect server units and, finally, objects (things) that generate data and carry out specific operations. In contrast to cloud computing, where a set of homogeneous resources are concentrated in the same area, fog computing infrastructure typically consists of a set of heterogeneous and geographically distributed resources, possibly organized in *fog regions* with respect to the objects each region hosts. Hence, in this context, workloads can be displaced from the central cloud to the edge, going through a continuum of device with heterogeneous capabilities in terms of processing, storage and networking, which are transparently made available by modern

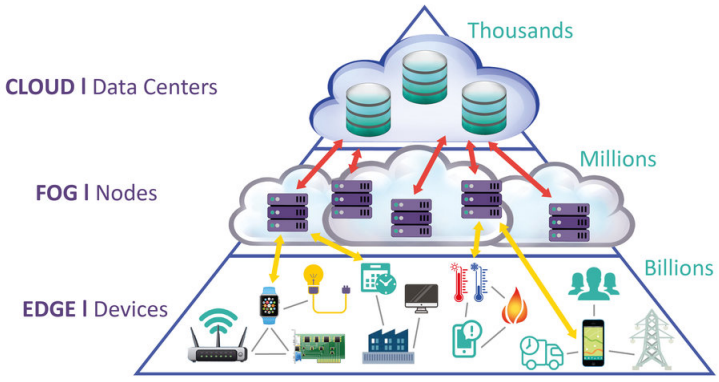

Figure 2.1: Example of Fog Architecture [1]

virtualization techniques [22]. The diversity of resources, their availability and location, are all factors that impact the capability of the infrastructure owner of successfully offering services with adequate quality. Given this heterogeneity, the main challenge comes from the need to deploy parts of each application on the central cloud or on the edge units, depending on the requirements of that particular application.

Fog Computing vs Edge Computing. Often, in the literature, the terms "Fog Computing" and "Edge Computing" are used interchangeably. However, according to the Industrial Internet Consortium [4], there is a marked difference between the two paradigms. Indeed, fog computing offers all possible services, including networking, computing and storage from the cloud to the things that generate data. Conversely, edge computing involves only the computation at the edge. A clear definition is given in [40]: "*fog is inclusive of cloud, core, metro, edge, clients, and things*" and '*fog seeks to realize a seamless continuum of computing services from the cloud to the things the network edges as isolated computing platforms*".

Business Challenges. Given the high level of heterogeneity along the continuum going from the cloud to the things, one main challenge is represented by the interaction between all the entities involved in this context. Indeed, although several companies are offering their *fog solutions* [2, 3], the problem is represented by the ownership of networking and computational resources along the road. For example, main resource providers, such as Amazon or Google, own huge amounts of computational resource but they have scarce control on last mile connections typically owned by telecommunication operators. Hence fog computing poses not only engineering problems but also business challenges related to agreements between the involved stakeholders.

However, the main challenge offered by the fog paradigm is represented by the complexity of management and orchestration of such a distributed computing infrastructure with respect to the traditional cloud computing environment. In this context, resource allocation, applications placement and orchestration must be conceived in a heterogeneous,

widespread scenario where locality, context-awareness and network performances must be taken into account [52].

2.2 Microservice Applications

The use of virtualization techniques, such as containers or virtual machines, on the infrastructure's resources simplifies the management of IoT services [39], coping with heterogeneity of IoT technologies [75].

With this new paradigm structure virtual machines or containers can run either in the central cloud, or over edge units, depending on the requirements of IoT applications. Actually, the current practice in cloud application design tends to favour microservice-based development, both for reasons of availability and of scalability. As reported in [15], IoT applications are not only monolithic but can consist of multiple interdependent components. With the fast growth of IoT, microservice-based development has become the current practice in cloud and fog application design in order to guarantee improved flexibility, availability and scalability [57, 108]. Microservice applications are composed by coupled modules, *e.g.,* a graphical user interface, a user repository, a web server, an image recognition module, or a monitor application. Usually, as depicted in Figure 2.2, some modules of an IoT application are connected to devices, such as cameras, acquiring some data to be processed by other modules. Once interconnected using a specific communication and computing pattern, the microservice architecture delivers the intended functionality while preserving scalability, minimality and cohesiveness of the application [57]. To this respect, it is natural to assume that fog-native applications should adhere to the modular microservice paradigm.

As an example, consider an application for number-plate-based car access control to a restricted traffic zone. The customer, *e.g.,* a municipality, requires an application that can access a well-located camera, convert the number plate from the video stream into text, store the number plate in a database and compare it with authorized number plates. Such a kind of application can be easily designed following a microservice-oriented architecture and would clearly benefit from video stream computation close to the camera, *e.g.,* to reduce bandwidth needs and improve privacy.

Thanks to this particular structure, applications can be represented through graph structures. Usually, such applications are represented as weighted DAGs (Directed Acyclic Graphs) where each node of the graph represents an application's component and an edge between nodes indicates a particular relation between components [103]. As shown in Figure 2.2, each node of the graph represents an application's component (microservice), and the edges represent some dependencies between components. The weights on the edges can represent some important metrics for the application, *e.g.,* the throughput generated between two microservices or the maximum tolerable delay on a specific application's link.

Recent studies have started to investigate the placement and load balancing problems

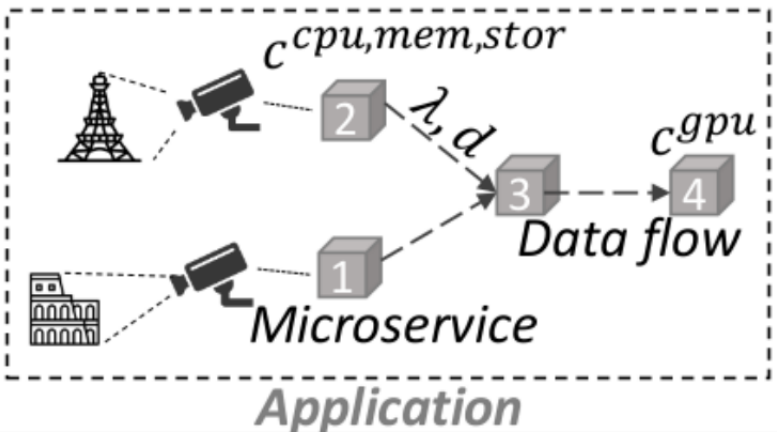

Figure 2.2: Example of microservice-based application

of such applications [116, 88] where each module has specific requirements in terms of computational and networking resources. Satisfying the requirements of all the modules of an application translates in the selection and allocation of the right set of resources in the fog-infrastructure. But, while resource allocation in cloud computing is a well-known and complex problem – provably NP-hard in all cases of practical relevance [86] – in fog computing the problem has a specific structure due to geo-distribution of heterogeneous resources and location of IoT devices. The placement of each application's module can depend upon the specific requirements of the modules and the objective of the network operator that is performing the placement.

2.3 Placement and Orchestration

Given the context of fog computing and the structure of IoT applications presented before, the placement and the orchestration of applications' microservices play a fundamental role for guaranteeing optimal performances in terms of cost for network operators, and Quality of Service (QoS) for users. Both these phases involve the use of resource allocation techniques deeply used in the cloud computing context [102]. Such an environment presents homogeneous resources where standard technologies are based on overprovisioned data centers [42, 68]. In fog computing, the business of edge infrastructure owners can not rely on overprovisioning. Rather, they need to trade off localized data processing and low round-trip time for storage, memory and processing capabilities of edge units [117, 103, 69]. In cloud computing the computation is limited to a few data centers with homogeneous and powerful computational capabilities and there are no needs for bandwidth optimization. On the other hand, in the fog computing context, the computation can be displaced among all the cloud-to-things continuum and it is distributed among different fog regions involving heterogeneous nodes for the computation. This also requires a careful optimization of the bandwidth usage between different regions.

The *placement* phase considers the initial deployment of all the applications modules on the infrastructure such that all the applications requirements are satisfied optimizing a given cost function. Each application can be distributed between the cloud and fog regions depending on the optimal allocation according to the specific cost function. Typically, cost functions involve the payment due to the use of external resources such as public cloud resources. Most of the placement problems we are dealing with are proved to be NP-hard preventing us to provide exact algorithmic solutions.

In Chapter 3 and Chapter 4 we introduce two different placement problems in two different contexts, and we provide discussions on the problems complexity and approximated solutions obtained by the application of heuristic methods.

On the other hand, the *orchestration* step involves the deployment of applications as their demands vary over time. Indeed, once the application demand changes, a decision on the resource allocation must be taken. The orchestration involves all the decisions regarding the resource allocation as the applications' demands change in order to guarantee the desired applications' requirements and to optimize specific cost functions at runtime. In Chapter 5 and in Chapter 6 we tackle two orchestration problems involving delay and budget constraints on the deployment of applications among the cloud and the fog.

2.3.1 Multidimensionality

It is worth noting that both the placement and the orchestration are dealing with multidimensional problems due to the different resource requirements such as CPU, storage and network capacity. As shown in [93], the multidimensional case is significantly different from the single dimension one. Multidimensional resource allocation schemes may vary depending on the solution space of the specific problem. In this thesis we can detect two main features that significantly influence the type and the solution space of the problem:

- *Application Structure*: applications can be considered fitting the most general form, *i.e.,* a graph structure where we have a set of interdependent modules, or, on the other hand, they can be considered as single entities consisting of single containers. However, each module has multidimensional requests representing their needs in terms of CPU, memory and storage. Furthermore, replication of different modules can be taken into account (autoscaling).

- *Infrastructure Scenario*: in this thesis we take into account two different scenarios. The first one is a *multi-servers* scenario where we have different servers distributed in several regions where to deploy microservice applications. The second type of scenario that we consider is a *single-server* scenario where the orchestration of applications between the cloud and a fog server is explored.

Obviously, different combinations of all the possible instantiations of the described features can give rise to different problems. In this thesis we cover most of these combinations. In particular, in Chapter 3 and in Chapter 4 we take into account the most

general scenario for both the application structure and the infrastructure, considering microservice graph-based applications and a general infrastructure with multiple servers distributed among different regions. In Chapter 5 we consider the orchestration problem with pipelines applications in a single server scenario. Finally, in Chapter 6 we tackle the problem of orchestration of single-container applications with replication possibility in a multi-server scenario.

Chapter 3

Throughput-aware Partitioning and Placement of Fog Applications

In this chapter we present the problem of applications placement in a heterogeneous, geographically distributed scenario such as the fog computing one. We assume the perspective of an infrastructure provider that aims to maximize her revenue while satisfying the requirements of microservice-based applications. We root our analysis on the throughput requirements of the applications while exploiting offloading towards different regions. An algorithmic solution is designed to optimize the placement of applications modules either in cloud or in fog. The overall solution consists of two cascaded algorithms: the first one performing a throughput-oriented partitioning of the applications' modules, and the second one rules the orchestration of applications over a region-based infrastructure. Extensive numerical experiments validate the performance of the overall scheme and confirm that it outperforms state-of-the-art solutions adapted to our context. This chapter is mostly based on works [49, 50].

3.1 Introduction

Whilst several approaches have been proposed in the cloud literature, few of them account for the deployment among different regions. In fact, existing technologies fit the cloud scenario where the resources' pool is concentrated in the same area irrespective of the objects' location. For instance, current Kubernetes' placement algorithms [5] perform well in cloud, but they require new functional extension to discriminate the deployment of application components across multiple regions. Consequently, there is a lack of solution for application deployment that exploits the inter-operability between different regions. Actually, in our tests we verify that, while for the sake of implementation it is tempting to treat all the fog regions as a unique region ruled by off-the-shelf Kubernetes placement algorithms, this may severely limit the number of applications deployed on the infrastructure.

3.1.1 Main Contribution

In this chapter we address the problem of an infrastructure provider whose aim is to deploy a batch of applications in a fog infrastructure covering different geo-distributed and heterogeneous regions, subject to the applications' requirements with respect to both computation and communication. We represent IoT applications' workflows by means of direct acyclic graphs (DAGs), where microservices are vertices and the dependencies between the microservices represent graph edges. The corresponding resources optimization problem results to be mixed integer non linear, and NP-hard. The problem combines a multicommodity flow and a graph embedding problem, for which we provide a negative result for the possibility to design tight polynomial-time approximation algorithms. Thus, we propose a cascade solution consisting of two main steps. First, at the application level, a preliminary partitioning for applications minimizes the throughput footprint of fog-native applications. Second, a placement step accounts for the computational and communication demands and proximity requirements of applications.

3.2 State of the Art

In cloud and mobile cloud computing, the problem of microservices applications deployment has been thoroughly studied. As described in [15], cloud software design privileges modular software structures where applications are composed by multiple coupled components known as microservices. In [95], applications are assumed to have a microservice architecture: the authors proposed a distributed mechanism for microservices scheduling at the edge. The objective is to minimize the service latency for all the applications to be deployed. However, applications are simply represented as sets of independent microservices. In [59], instead, microservice fog applications are represented as DAGs. With such an application structure, the general application deployment problem bears several similarities with the graph embedding problem [37, 27], a well-known NP-hard problem. In our context, even for two-module partitioned fog-applications, such a problem is proved to still be NP-hard. In [116], a DAG structure for IoT applications similar to the one used in this chapter is considered. However, the aim of the work is different from ours, since the objective there is to perform communication load balancing among the microservices of a single application via their replication across the infrastructure. Furthermore, the infrastructure model is characterised only by the nodes that host some replicas of the application's microservices.

In [117], application provisioning is studied from the perspective of the network infrastructure. A fully polynomial time approximation scheme is derived for single and multiple applications deployment, showing significant QoS performance improvement with respect to applications' bandwidth and delay figures. However, applications are represented as a single module communicating with a set of IoT devices generating data. For this reason, authors focused mostly on application's networking requirements.

Taneja et al. [103] defined a placement algorithm by mapping the DAG of the modules of an IoT application into fog and cloud nodes. Numerical results show performance gains in terms of latency, energy and bandwidth constraints, compared to edge-agnostic placement schemes. Our work, conversely, develops an optimization framework able to account for both traffic and computing demands of a whole batch of applications to be deployed over multiple regions.

The DAG-like applications deployment on an infrastructure network reminds the *Virtual Network Embedding* problem, a well-known NP-hard problem deeply studied especially in the topic of Virtual Network Functions (VNFs) placement [37]. Many heuristic solutions have been proposed in the literature [33] since the general problem presents strong inapproximability results [14]. The main difference with respect to our context is that, usually only CPU constraints are taken into account in VNFs problems. Conversely, in a fog scenario, where resources are limited and heterogeneous, all the requirements (CPU, memory and storage) of each microservice should be considered. Furthermore, the network infrastructure where VNF applications are deployed is usually not partitioned into regions.

Partitioning the applications' computation process represents a promising technique to guarantee high performance in mobile cloud or edge computing, especially for inference tasks with deep neural networks. E.g., Pachecom et al. [90] presented a partitioning algorithm, based on the shortest path problem, for the layers of deep neural networks to reduce the inference delay of a BranchyNet [104] distributed between the edge and the cloud. The authors of [111] studied the problem of computation partitioning with the aim of maximizing the application throughput in processing the streaming data. A genetic algorithm is able to find the best partition in between the cloud and the mobile at runtime. In [47], a general technique to minimize the execution time of IoT applications is proposed. The model introduced takes into account computation and communication delays. By reduction to the Matrix Chain Ordering Problem, an algorithm is provided in order to solve the optimization problem via dynamic programming, with time-complexity log-linear in the number of operators of the application. With respect to such solutions, we have different objective, that is to maximize the infrastructure owner's revenue (maximizing the number of successfully deployed applications) by combining efficient applications' partitioning while avoiding network bottlenecks.

One of the novelties of this work is the multi-regions scenario for the applications partitioning and placement. With respect to the container technologies discussed in this work, the de-facto standard for container orchestration is Kubernetes [29] even if new technology solutions for the edge are being proposed [6]. As we will describe in the next sections, resource allocation in Kubernetes proceeds by first enlisting servers able to host a target application module in a container pod. In the native cloud version, the actual container deployment is agnostic of the notion fog region and agnostic of network conditions. Our placement logic, instead, is able to address also the locality of object

Table 3.1: Main notation used throughout the chapter

Symbol	*Meaning*
$\mathcal{K}$	set of regions $\lvert\mathcal{K}\rvert = K$
$\mathcal{U}$	set of applications to be deployed $\mathcal{U} = \cup_{i=1}^{K} U_i$, $\lvert\mathcal{U}\rvert = U$
S_k	set of server units in region k, with $\lvert S_i\rvert = n_i, \forall i \in \mathcal{K}$, $S_i = \{s_{i_1}, \ldots, s_{i_{n_i}}\}$
S_0	central cloud
U_k	set of applications requiring IoT data in region k
F_u	output samples per second required by application u
$\mathbf{C_{k_i}}$	memory, storage and processing capacity of the i-th server in region k: $\mathbf{C_{k_i}} = (C^M_{k_i}, C^S_{k_i}, C^P_{k_i})$
$\mathbf{c_m}$	memory, storage and processing requirements of microservice a m: $\mathbf{c_m} = (c^M_m, c^S_m, c^P_m)$
$x_{u,k,i} \in \{0, 1\}$	boolean variable indicating the fog chunk of application u is placed on server unit i of region k
$x_{u,k}$	$x_{u,k} = \sum_{i \in S_k} x_{u,k,i}$
$x_{u,u}$	boolean variable indicating the fog chunk of application u is placed on the same region of the IoT object requested by the application u

demands and their cumulative effect onto the communication infrastructures.

3.3 System Model

The combination of a fog-native partitioning and a placement scheme follows the rationale that microservices of the same application, when deployed on the same fog region, generate negligible communication overhead; in fact, under standard containerisation technologies they can be deployed on the same pod [29, 7]. On the other hand, when two application modules are deployed in different region, the resources allocation balance must account for the communication overhead. A rational choice is to group applications microservices according to their requirements: fog modules with strict latency constraints require installation on the edge, *e.g.,* to support real-time processing of data streams from a local IoT device. Other microservices may need computational power not available on edge nodes, this is the case, *e.g.,* of online machine learning algorithms. This fog-cloud dichotomy greatly simplifies the placement of such fog-native applications, since while the first group hosts microservices that could be or need to be deployed on the edge, the second one is the set of microservices without strict latency requirements and, in turn, possibly hard computational requirements; for the latter, cloud deployment is assumed apriori. This minimal partitioning results in the functional bi-partition shown in Figure 3.1.

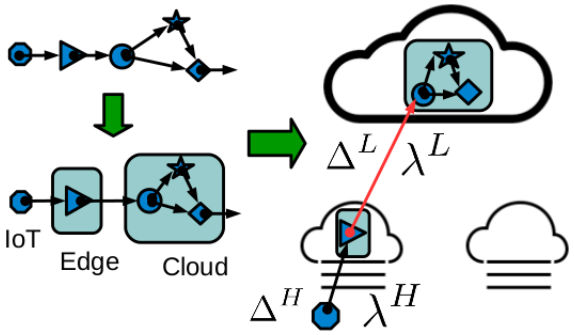

Figure 3.1: Cascade of throughput-aware partitioning of a fog application into two chunks, and subsequent placement of the fog chunk onto an edge node.

3.3.1 Network Model

We consider a standard fog system architecture [91, 21] where the network architecture consists of a central cloud, which is assumed of unlimited computational capabilities – relatively to edge nodes – and a set of fog regions connected to the central cloud. Each fog region comprises servers with specific computational capabilities in terms of memory, CPU and storage. We consider a batch of applications to be deployed on such an infrastructure. Each application is described by a list of requirements in terms of memory, CPU, storage and bandwidth. The objective is to deploy the applications on the infrastructure in order to maximize a certain profit function while satisfying the resources constraints and applications' demands. Throughout the chapter the target performance figure is the number of concurrent fog-applications hosted simultaneously on the infrastructure; more general objective functions can be studied in the same framework and are left as part of future works. *The problem is to find a set of mappings of applications onto fog-regions to maximize such objective function, where each such mapping engenders diverse resource occupation vectors and its own profit value.*

More formally, we consider a fog system deployed over a set of geographic regions $\mathcal{K} = \{1, \ldots, K\}$. Region k hosts a set S_k of edge servers or units. We denote s_{k_i}, with $i \in \{1, \ldots, n_k\}$ a specific edge unit deployed within the k-th region; for the sake of notation we denote the central cloud as S_0. The resources of edge unit s_{k_i} are represented by capacity vector $\mathbf{C}_{k_i} = (C^M_{k_i}, C^P_{k_i}, C^S_{k_i})$. The first component of the capacity vector is the memory capacity, while the second component is the processing capacity; the third component denotes the storage capacity, *i.e.,* the data volume that can be accommodated on the storage of the edge unit. The fog infrastructure can be described by an undirected weighted graph $G = (V, E)$ where $V = \{S_i \cup U_i\}_{i \in \mathcal{K}}$ and $E \subseteq \{\{i, j\} | i, j \in V, i \neq j\}$. The weight of each edge $\{i, j\} \in E$ consists of the latency on the link d_{ij}, and the link bandwidth B_{ij}. Let $\mathcal{N}(S_i) = \{S_j | \{j, i\} \in E\}$. Figure 3.2 reports a pictorial example of such fog infrastructure.

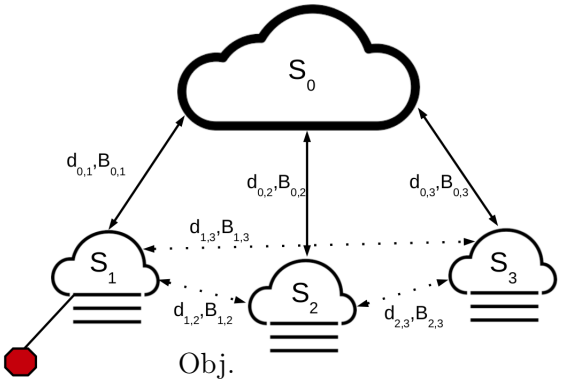

Figure 3.2: Reference fog system architecture: application u_1 requests data from an IoT object (marked red) located in area S_1; the whole system is composed of geographical areas (regions) $S_i, i = 1, 2, 3$ connected to the central cloud S_0.

3.3.2 Application Model

The application model adopted in this chapter is based on the distributed data flow model [103], where distributed components of an IoT application perform different computational tasks. We model each application like a weighted DAG where the nodes represent the application modules and the edges between nodes represent the execution order of the modules. The weight of each edge represents the throughput generated by one microservice performing some computation and sending the results to another component downstream. More formally, each application u is defined by the weighted digraph (G_u, λ_u), where $G_u = (V_u, E_u)$ and $\lambda_u : E_u \to \mathbb{R}^+$, *i.e.,* $\lambda_u(m, n)$ represents the maximum throughput that can be generated by the microservice m and sent to microservice n of the same application. Each microservice m is characterized by its requirements $\mathbf{c_m} = (c_m^M, c_m^S, c_m^P)$ in terms of memory, storage and processing. In each fog region k we have a set of applications to be deployed, U_k. From here on out, we identify the application and the device from which data are requested with same symbol. We say that application u "belongs" to a given region because the IoT object is located there. Such region is denoted S_u for the sake of notation. Furthermore, we denote with $\mathcal{U} = \cup_{i=1}^{K} U_i$ the set of all applications to be deployed on the infrastructure. A further assumption that we shall introduce is that each application is deployed on a set of servers defined by the neighbourhood of the region from which the data are generated, as in Figure 3.2. To this respect, we remark that the practice to ensure connectivity between containers is to interface them at the application layer, *e.g.,* connecting them by http(s) protocol, so that multi-hop routing at the network layer is not viable for joint optimization as it would introduce queuing and routing delay at each hop [97]. In the proposed system model, placement on neighbouring regions still attains load balancing while greatly simplifying system design. Nevertheless, in the numerical section we perform a thorough comparison

of our reference system with other solutions violating such assumption.

3.3.3 Problem Formulation

The **main problem** can be resumed as follows:

Given a set of DAG-like applications $\mathcal{U}$ and a fog infrastructure $G = (V, E)$, deploy the maximum number of applications in order to maximize the provider's revenue while satisfying the applications' delay requirements and the infrastructure capacity constraints.

The deployment of an application $u \in \mathcal{U}$ is represented by a mapping between each module $v \in V_u$ and a server of the infrastructure such that all the applications requirements and the infrastructure constraints are satisfied. These constraints involve the CPU, memory, storage and bandwidth constraints for the infrastructure, and delay constraints for the application. This latter constraint involves the computation of the optimal throughput for each edge of the application $(v, w) \in E_u$ given the placement of application's nodes v and w. In order to compute such throughput values we should take into account the maximal path, in terms of latency, from the source, the region where v has been deployed, to the destination node, the region where w has been deployed. This would result in having a non-linear constraint involving binary variables for the placement of each application's module and continuous variables for the computation of the optimal throughput for each application's edge.

Main Problem Formulation

Here we provide a formal description of the main problem.

Variables. We introduce a binary variable for the placement of each applications' microservice on a single server of the infrastructure:

$$x_{k,s}^{u,m} = \begin{cases} 1, & \text{if microservice } m \text{ of application } u \\ & \text{is placed on server } s \text{ of region } k \\ 0, & \text{otherwise} \end{cases}$$

$\forall u \in \mathcal{U}, \forall m \in V_u, \forall k \in \mathcal{K} \cup \{0\}, \forall s \in S_k$.

For the applications' links mapping to the physical paths of the infrastructure we introduce a second kind of binary variables:

$$y_p^{u,(m,n)} = \begin{cases} 1, & \text{if } (m,n) \in E_u \text{ is mapped to } p \in \mathcal{P} \\ 0, & \text{otherwise} \end{cases}$$

$\forall u \in \mathcal{U}, \forall (m,n) \in E_u, \forall p \in \mathcal{P}$, where $\mathcal{P}$ is the set of all paths between nodes in the infrastructure graph G.

Furthermore, since the objective of the problem is to maximize the number of applications entirely deployed on the infrastructure (*i.e.,* maximizing the revenue), we introduce

a third binary variable to indicate whether an application is entirely deployed on the infrastructure:

$$z_u = \begin{cases} 1, & \text{if application } u \in \mathcal{U} \text{ is entirely deployed} \\ 0, & \text{otherwise} \end{cases}$$

$\forall u \in \mathcal{U}$. Hence, z_u will be set to 1 if and only if all the microservices and links of application u are deployed on the infrastructure.

Finally, we have continuous variables to control the optimal throughput generated on applications' links in order to satisfy the delay constraints:

$$\lambda_u(m,n) \in \mathbb{R}^+, \quad \forall u \in \mathcal{U}, \forall (m.n) \in E_u.$$

Constraints. First, we have resource capacity constraints for each fog server of the infrastructure:

$$\sum_{u \in \mathcal{U}} \sum_{m \in V_u} \mathbf{c_m} x_{k,s}^{u,m} \leq \mathbf{C_{k_s}}, \quad \forall k \in \mathcal{K}, \forall s \in S_k, \tag{3.1}$$

where $\mathbf{c_m}$ and $\mathbf{C_{k_s}}$ are the requirements of microservice m and the capacities of server s of region k, respectively, in terms of memory, storage and processing.

Then we have all the constraints related to the applications' links mapping:

$$\sum_{p \in P_{k,k'}} y_p^{u,(m,n)} = \sum_{s \in S_k} x_{k,s}^{u,m} \wedge \sum_{s \in S_{k'}} x_{k',s}^{u,n}, \tag{3.2}$$

$\forall (k,k') \in V \times V$, $\forall u \in \mathcal{U}$, and $\forall (m,n) \in E_u$. Constraint (3.2) ensures that a unique physical path of the network infrastructure is used by an application link whenever the application nodes connected by such link are deployed on the extreme nodes of the physical path (the symbol $\wedge$ represents the Boolean *"and"* operator). Note that the sums on the right side of the equation (3.2) are always either 0 or 1 since, as shown later, each microservice can be deployed on at most one location, *i.e.*,, one server of a certain region.

We have to add constraints to guarantee a unique placement for each component of each application (microservices and links):

$$(\sum_{m \in V_u} \sum_{k \in V} \sum_{s \in S_k} x_{k,s}^{u,m} = |V_u|) \vee (\sum_{m \in V_u} \sum_{k \in \mathcal{K}} \sum_{s \in S_k} x_{k,s}^{u,m} = 0), \quad \forall u \in \mathcal{U}, \tag{3.3}$$

$$\sum_{k \in V} \sum_{s \in S_k} x_{k,s}^{u,m} \leq 1, \quad \forall u \in \mathcal{U}, \forall m \in V_u. \tag{3.4}$$

$$\sum_{p \in \mathcal{P}} y_p^{u,(m,n)} \leq 1, \quad \forall u \in \mathcal{U}, \forall (m,n) \in E_u. \tag{3.5}$$

Constraint (3.3) guarantees that either all the applications' microservices are deployed or no one of them is deployed (the symbol $\vee$ is the Boolean *"or"* operator). Note that the

two conditions are mutual exclusive so they cannot be true at the same time. The same thing must be guaranteed for the applications' links:

$$(\sum_{(m,n)\in E_u}\sum_{p\in\mathcal{P}} y_p^{u,(m,n)} = |E_u|) \vee (\sum_{(m,n)\in E_u}\sum_{p\in\mathcal{P}} y_p^{u,(m,n)} = 0). \tag{3.6}$$

An additional constraint should be added to guarantee a complete deployment for each application. If all the microservices of an application are deployed then all the application's links must be mapped to a physical path and vice versa.

$$\sum_{m\in V_u}\sum_{k\in V}\sum_{s\in S_k} x_{k,s}^{u,m} = |V_u| \Leftrightarrow \sum_{(m,n)\in E_u}\sum_{p\in\mathcal{P}} y_p^{u,(m,n)} = |E_u|. \tag{3.7}$$

The following are constraints on the bandwidth capacity for each physical link $(k, k') \in E$:

$$\sum_{u\in\mathcal{U}}\sum_{(m,n)\in E_u}\sum_{p\in P:(k,k')\in p} \lambda_u(m,n)\, y_p^{u,(m,n} \leq B_{kk'}. \tag{3.8}$$

Furthermore, we have constraints that bind x and y variables to z for each application $u \in \mathcal{U}$:

$$z_u = \left\lfloor \frac{\sum_{m\in V_u}\sum_{k\in K}\sum_{s\in S_k} x_{k,s}^{u,m}}{|V_u|} \right\rfloor \wedge \left\lfloor \frac{\sum_{(m,n)\in E_u}\sum_{p\in\mathcal{P}} y_p^{u,(m,n)}}{|E_u|} \right\rfloor. \tag{3.9}$$

Indeed, for each application $u \in \mathcal{U}$, the decision variable z_u is set to one if and only if all the application's modules and links are deployed.

Finally, we have delay constraints for each application:

$$\max_{p_u\in P_u}\left\{\sum_{(m,n)\in p_u}\sum_{p\in\mathcal{P}}\left(\frac{\Delta_{m,n}^u}{\lambda_u(m,n)} + y_p^{u,(m,n)}\sum_{(k,k')\in p} d_{k,k'}\right)\right\} \leq \frac{1}{F_u}, \tag{3.10}$$

where P_u represents the set of all directed paths between the source and the destination node of the application u, and $\Delta_{m,n}^u$ is the data transmitted from the module m to module n of the application u.

Objective. The objective function is the revenue of the infrastructure's owner based on the number of applications entirely deployed:

$$\text{maximize} \sum_{u\in\mathcal{U}} f_u\, z_u \tag{3.11}$$

Complexity and Approximability of the main problem

Regarding the computational complexity of the main problem, it can be observed that - by fixing throughput variables - the problem can be proved to be NP-hard by polynomial reduction from a virtual network embedding problem, known to be strongly NP-hard.

Indeed, as it can be noticed from the formulation, the main problem may resemble a Virtual Network Embedding (VNE) problem with in addition the decision variables and constraints for the throughput on the applications DAGs edges. Overall the main problem appears non-linear. Furthermore, results in [14] show that the VNE problem is strongly NP-hard with inapproximability results obtained by reduction from the Maximum Stable Set Problem (MSSP). Thus, unless P = NP, no polynomial time approximation scheme can be found within a factor of $n^{\frac{1}{2}-\epsilon}$ for any $\epsilon > 0$, where the term n represents the total number of servers available for microservices' deployment: we have n_i servers for each region $i \in \mathcal{K}$, plus the cloud region which counts as a single server with infinite capacity.

Proposition 3.1. *Unless P = NP, the application placement problem can not be approximated in polynomial time within a factor of $n^{\frac{1}{2}-\epsilon}$ for any $\epsilon > 0$, where $n := 1 + \sum_{i=1}^{K} n_i$.*

Proof. The proof holds by reduction from the VNE problem, as defined in [14], to the main problem. Let assume that the throughput variables on the applications' edges are fixed. Given the substrate graph $G^0 = (V^0, E^0)$ of VNE problem and the set of requests R, where each $r \in R$ is a graph $G^r = (V^r, E^r)$, we map each node in V^0 to a server of the fog infrastructure $G = (V, E)$ and we map all each request $r \in R$ to an application with V^r microservices and E^r edges. The nodes and edges capacities of G^0 are mapped to servers capacities and to the bandwidth capacities of the links in G, respectively. The demand for each node and the traffic demand of each edge in G^r are mapped to microservices requests and edges' throughputs of each application. Exploiting the transitivity of polynomial reduction, Corollary 3.3 in [14] applies to our problem. □

3.3.4 Resolution Approach

The previous result rules out the possibility to devise efficient approximation algorithms to solve the main problem altogether, in all cases of practical interest. Thus, in order to render the original problem more tractable, we split it into the cascade of two sub-problems. In the first one, each application is partitioned in two chunks in order to minimize the maximum throughput between them, and imposing that all the application's modules belonging to the same chunk will be deployed in the same region of the infrastructure. In this manner, the communication overhead between application's components and the number of binary and continuous variables for the placement and throughput computation are reduced. In the second sub-problem, once fixed the partition for each application solving the previous problem, a feasible deployment per application's chunk is found. The latter problem will be further transformed into an integer linear programming problem.

As introduced before, after partitioning step, each fog application will consist of two chunks: the first one is the subset of application microservices apriori executed in cloud, whereas the second is the subset encompassing microservices that can be deployed either in fog or in cloud. By assumption, all applications are supposed to adhere to such a

fog-oriented *partitioning*[1]. Of course, in general, any partition of microservices into two chunks will do. However, we shall consider both the case of throughput-agnostic and that of throughput-aware partitioning. With the former, the application partition is oblivious to the data flow across microservices of the same application. In the latter case, instead, we shall optimize partitioning based on the communication requirements of the microservice architecture. In both case, anyhow, it is reasonable that a fog-native partitioning would be completely infrastructure-agnostic, thus accounting for requirements of a tagged application only.

Note that the partitioning phase can be realised by a static and offline algorithm processing each application individually. Depending on the business model, the infrastructure provider can offer the application owner with a cloud service to containerize her applications into a fog and a cloud chunk using the proposed algorithm. For the sake of simplicity, we have omitted all the aspects related to the orchestration of the registries where the images of the chunks are finally stored . In fact, multiple schemes can be envisaged, *e.g.,* either a a single cloud registry, or also several per region registries caching application chunk images. Once the registries have been populated with the chunk images of the applications concerned with target areas, the placement algorithm could leverage, *e.g.,* the Kubernetes control plane to switch on the fog chunks of applications in the target regions. How to optimize the registry placement and the chunk image caching process will be discussed in next chapters.

The cloud computing literature provides well-known solutions in a single data-center, such as, for instance, Kubernetes' scheduling algorithms [5]. Although these algorithms show good results in a single-region scenario, they are not designed to perform under throughput constraints among several fog regions. Our target is an algorithmic solution identifying a region for the deployment of each partitioned application. Focusing on the region selection only complies with the current practice in containers' orchestration, since Kubernetes orchestration procedures and/or optimized versions can be later applied within each region to find the best local resource allocation. Overall, the complete placement scheme proposed consists of the following logical phases which we describe next as described in Figure 3.1:

1. *Fog Application Partitioning*: application packaging into a cloud and fog chunk.

2. *Region Selection*: selection of target regions in the neighbourhood of the objects that generate data consumed by fog applications.

3. *Deployment*: using an off-the-shelf orchestrator (*e.g.,* Kubernetes) to perform allocation within a fog region cluster according to standard or optimized local placement rules.

[1]Depending on the virtualisation technology, a fog-oriented application partitioning will produce two containers/pods or two VM images; however this is not relevant for the rest of our discussion.

3.3.5 Throughput-aware fog partitioning

An efficient bipartition of the application graph can reduce monitoring and networking costs. Indeed, for the sake of example, we can consider the case of video surveillance applications. We can expect one or more of the application microservices to require significant amount of computational power in order to execute, *e.g.,* sophisticated face-recognition techniques. Such microservices will be normally part of the cloud chunk. On the other hand, since the throughput generated by video sensors represents a serious bottleneck, we can also expect some filtering microservice to pre-process the raw video stream on edge nodes, *e.g.,* to extract just the frames relevant for the task from the raw video stream. Thus, in order to reduce upstream the fog-to-cloud throughput of such an application, it is indeed optimal to install, whenever possible, such a microservice in the fog chunk.

Overall, an efficient throughput-aware partitioning shall split microservices such as to minimize the total throughput flowing from the first set of the microservices partition, the fog chunk, to the second set of the partition, the cloud chunk. Let the weighted DAG $G_u = (V_u, E_u)$ represent the target application u. The application partitioning problem can be reduced to the *Minimum k-cut* problem on graph G_u. The standard definition for a *k-cut* is set of edges whose removal leaves k connected components [106]. A minimum k-cut problem asks for a minimum weight k-cut. In our context, we are looking for a minimum 2-cut on G_u. The reason behind such a minimal cut is twofold. First, since we are not assuming a hierarchical structure for the infrastructure, only fog regions and the cloud can perform computational tasks. The IoT devices, indeed, are not usually able to perfotm significant operations. Furthermore, within the same region, delay and bandwidth constraints are negligible meanwhile they are relevant between two different regions. Hence, splitting an application in more than two chunks would be equivalent to distribute the applications in more than two regions and this would bring additional communication overhead in terms of latency and routing implementation issues for the applications. Finally, two different regions may not be directly connected forcing the chunks of the applications to communicate through other regions. However, in this case we would lose what we have gained in terms of latency with fog computing. Secondly, if we wanted to partition an application in more than two chunks, we should solve a minimum k-cut problem with $k \geq 3$ and the partitioning problem would be NP-hard [106].

Given a source and a destination node, the *minimum 2-cut* problem is equivalent to the *maximum flow* problem for the *min-cut max-flow* theorem [43] which can be efficiently solved by a maximum flow algorithm such as the Ford-Fulkerson algorithm [55]. Hence, solving an *s-t* cut problem we can solve our application partitioning problem as expressed formally by Proposition 3.2 where APP-PARTITIONING is the application partitioning problem described above and MIN-CUT is the *minimum s-t cut* problem.

Proposition 3.2. *APP-PARTITIONING* $\leq_p$ *MIN-CUT.*

Algorithm 1: Application Partitioning Algorithm.

Input: Application graph $G_u = (V_u, E_u)$, $C \subseteq V_u$
Output: 2-cut partition of the application u

```
1  Cloud ← ∅;
2  Edge ← ∅;
3  V_u ← V_u ∪ {t};
4  for v ∈ C do
5  |   E_u ← E_u ∪ {(v, t)};
6  |   λ_u(v, t) ← +∞;
7  s ← first node in a topological order of G_u;
8  G'_u ← Max_Flow(G_u, s, t, λ_u);
9  for w reachable from s in G'_u do
10 |   Edge ← Edge ∪ {w};
11 Cloud ← V_u \ Edge;
12 return (Edge, Cloud)
```

Proof. We can show the polynomial reduction by taking an application DAG and selecting a source and a destination node (since we have a DAG structure there exists at least one node with no incoming edges, the source, and one node with no outgoing edges, the destination). Additionally, we set the maximum throughput of each application's edge as the maximum capacity of the graph. In this manner, by solving the min-cut problem on such graph we have a 2-cut partitioning of our application graph. □

Incidentally, whenever a single source and sink node pair can not be identified in the application DAG, it is always possible to add a virtual source (sink) node adding large-capacity edges and apply the algorithm to the resulting DAG.

The partitioning algorithm is described in Algorithm 1. The input is represented by the application graph and a subset C of application's microservices to be deployed in cloud by default. The algorithm adds a dummy target sink node t to the application graph; t is connected to each node in C using a dummy high-throughput edge. First, the algorithm establishes a source node through a topological sort of the application's DAG. Second, the Ford-Fulkerson's algorithm, applied to the augmented graph, returns the partition separating s and t with a cut of minimum capacity. All the nodes still reachable from the source node in the residual graph G'_u will be included in the `Edge` set, while the others will go to the `Cloud` set. We observe that the high throughput of the edges between nodes in C and node t grants that all nodes in C will belong to the `Cloud` set, as stated by Lemma 3.1.

Lemma 3.1. *For each node $v \in V_u \setminus \{s\}$, if $v \in C$ then $v \in$ `Cloud` when Algorithm 1 terminates.*

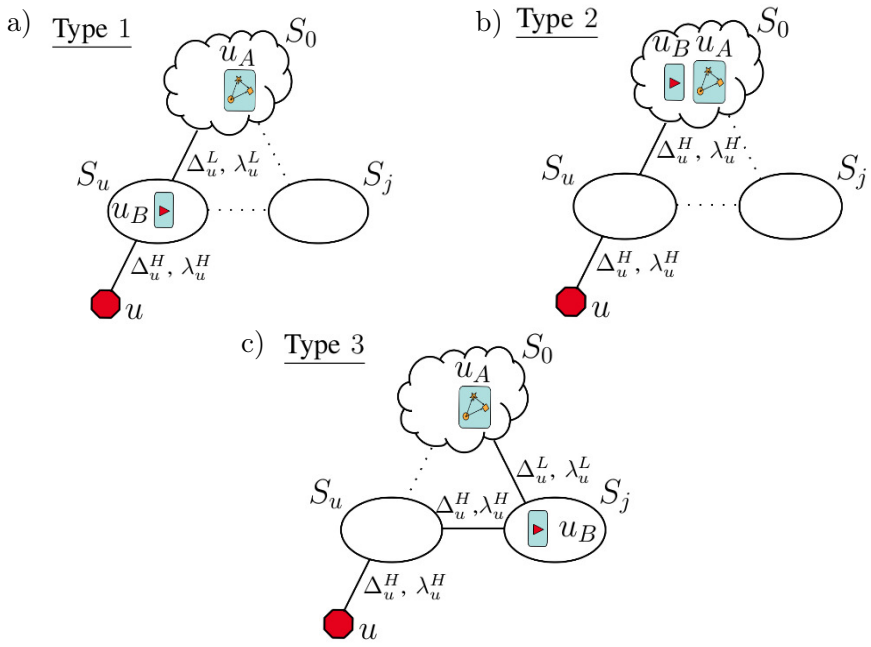

Figure 3.3: Configurations types for the deployment of the `Edge` module u_B; `Cloud` module u_A is always installed in cloud.

Proof. Let assume, by contradiction, that for some $v \in C$ it holds $v \notin$ `Cloud`. Since, by assumption, $v \neq s$, there exists an in-going edge to v, (v', v). If $v' \notin C$, then $\lambda_u(v', v) < \lambda_u(v, t)$, contradicting the cut minimality found by the Algorithm 1. If $v' \in C$ we can repeat the previous argument until we find the first node $v'' \notin C$, in the worst case until reaching node s. □

Complexity. The complexity of Algorithm 1 is dominated by the time complexity of the Ford-Fulkerson's algorithm which is $O(|E_u|\,\lambda^*)$ for each application u, where λ^* is the maximum flow in the application network. We can improve the runtime through the Edmonds-Karp implementation of the algorithm [45, 46] obtaining a complexity of $O(|V_u| \cdot |E_u|^2)$ independent of the maximum flow of the application network.

Throughput-intensive applications

Notation. From here on out, for the sake of brevity, for each application $u \in \mathcal{U}$, we identify with u_A and u_B the `Cloud` and the `Edge` chunks, respectively. Finally, let denote c_u^M, c_u^S, c_u^P the total resource requirements, in terms of memory, storage and processing capacity, respectively, of u_B; with compact notation we denote $\mathbf{c_u} = (c_u^M, c_u^S, c_u^P)$.

While the proposed framework does apply to other classes of applications, *e.g.,* memory-intensive or CPU intensive applications, we shall focus on optimizing the allocation of resources for *throughput-intensive* fog applications. They are a critical class of IoT applications which includes, *e.g.,* streaming mining applications [31], where processing directly

on fog nodes leads to huge bandwidth savings. In this context, for instance, a raw video stream can be filtered to extract only the relevant parts of specific frames, *e.g.,* those containing a face which need to be sent to the cloud to perform recognition. Hence, after the partitioning step showed in Figure 3.1, two different data units may be transmitted. We call, for each application u, Δ_u^H the data unit transmitted directly from the IoT sensor (*e.g.,* video camera frames) to the Fog chunk. Δ_u^L is the data unit transmitted from the Fog chunk to the Cloud chunk once a processing step has been performed (*e.g.,* face images).

Delay Constraints. These two different data units are sent at different rate; we denote λ_u^H as the raw throughput generated by the IoT device and λ_u^L for the throughput generated towards the Cloud chunk once the Fog chunk of the application has processed the stream. In this case, typically $\lambda_u^H \geq \lambda_u^L$. In the fog resource allocation problem described in the next subsection the throughput λ_u^H and λ_u^L are decision variables which can be optimized in order to meet the applications' delay requirements. In fact, we can assume that each application u has to output every $1/F_u$ seconds a result like a positive or negative face recognition match. The Cloud chunk is installed in the central cloud S_0. We can hence consider the whole processing chain involved by the two-chunks and the related data transmission delay. We should also include the processing delay d_u of application u (if deployed back to back to the IoT object), plus the communication delay d_{uj}, which is the additional delay to retrieve data from region where the sensor belongs, when the Fog chunk is installed in region j. In the resource allocation problem we need to consider the processing and transferring time. Actually, the processing time for each information unit depends on the throughput between application chunks. Any application placement has to guarantee the application to process an information unit Δ_u in $\frac{1}{F_u}$ seconds. Thus, the allocation of such throughput depends on the application deployment configurations. Since the `Cloud` chunk is always installed on the central cloud, the three basic fog configurations to deploy application u are as in Figure 3.3:

- *Type 1:* Fog chunk deployed on S_u; higher throughput λ_u^H flows between IoT object u and region S_u, with IoT data unit Δ_u^H. Δ_u^L served between S_u and S_0 with low throughput λ_u^L;
- *Type 2:* Fog chunk deployed on central cloud S_0; IoT data Δ_u^H is served between S_u and S_0 with high throughput λ_u^H;
- *Type 3:* Fog chunk deployed on a neighbouring fog region $S_j \neq S_u$; lower throughput required between S_j and central cloud S_0. However, the IoT data $\Delta_u = \Delta_u^H$ is served between S_u and S_j with high throughput λ_u^H.

3.3.6 Fog Resource Allocation Problem

The objective of the infrastructure owner is to maximize the revenue obtained by provisioning her fog infrastructure to applications tenants. By using traditional scheme of pay

per use, she can settle a cost for each application deployment. A tenant owning u pays $f_{u,k} > 0$ euros per application installed in region k.

The objective is thus to host containerized fog applications such in a way to maximize the owner revenue, while satisfying the applications' requirements. We can obtain the optimal reward for a given batch of applications.

Decision variables $x_{u,k,i}$ are boolean variables indicating the placement of the fog chunk of application u on the i-th server of region k. Further, decision variables $\lambda_u^H, \lambda_u^L \in \mathbb{R}^+$ represent the optimal throughput in the large and small data unit transfer mode for application u, respectively. Indeed, the throughput declared for each application's link in the partitioning step is an upper bound on the actual throughput generated on each link of the application. The optimal allocation policy using a mixed integer non linear program (MINLP) is shown in (3.12). We let $x_{u,k} = \sum_{i \in S_k} x_{u,k,i} \quad \forall (u,k) \in \mathcal{U} \times \mathcal{K}$ for notation's sake. The objective function is the revenue gained by the infrastructure owner. The constraint (3.13) is meant component-wise: it bounds the resources utilization on fog servers in terms of memory, processing and storage capacity, respectively. Also, (3.14) and (3.15) bound the throughput generated by applications with respect to links' capacity. (3.14) accounts for all traffic from region k to the central cloud, whereas (3.15) accounts for the throughput across adjacent regions as in Figure 3.3c. By constraint (3.16), the total transmission and computing time needs to be smaller than the service rate of the application. We assume that, according to (3.17), each application has at most one deployment region, since, given the limited resources, it is not always possible to deploy all the applications. In particular, (3.18) indicates that each application can be deployed only on neighbour regions or on its original region.

$$\text{maximize:} \quad \sum_{(u,k)\in\mathcal{U}\times\mathcal{K}} f_{u,k}\, x_{u,k} \tag{3.12}$$

subject to:

$$\sum_{u\in\mathcal{U}} \mathbf{c_u}\, x_{u,k,i} \leq \mathbf{C_{k_i}}, \quad \forall k \in \mathcal{K}, \forall i \in S_k \tag{3.13}$$

$$\sum_{u\in U_k} (x_{u,k}\, \lambda_u^L + x_{u,0}\, \lambda_u^H) + \\ + \sum_{j\in\mathcal{N}(S_k)} \sum_{v\in U_j} x_{v,j} \lambda_v^L \leq B_{k0}, \quad \forall k \in \mathcal{K} \setminus \{0\} \tag{3.14}$$

$$\sum_{u\in U_k} x_{u,j}\, \lambda_u^H + \sum_{u\in U_j} x_{u,k}\, \lambda_u^H \leq B_{kj},\ \forall jk \in E, j,k \neq 0 \tag{3.15}$$

$$d_u + \frac{\Delta_u^H}{B_u} + \left(d_{uj} + \frac{\Delta_u^H}{\lambda_u^H} + \frac{\Delta_u^L}{\lambda_u^L}\right) x_{u,j} + \\ \left(d_{u0} + \frac{\Delta_u^H}{\lambda_u^H}\right) x_{u,0} + \left(d_{u0} + \frac{\Delta_u^L}{\lambda_u^L}\right) x_{u,u} \leq \frac{1}{F_u} \\ \forall u \in U, \forall j \in \mathcal{N}(S_u) \tag{3.16}$$

$$\sum_{k\in\mathcal{K}} x_{u,k} \leq 1 \quad \forall u \in \mathcal{U} \tag{3.17}$$

$$\sum_{k\in\mathcal{K}\setminus\{\mathcal{N}(u)\cup\{u\}\}} x_{u,k} \leq 0 \quad \forall u \in \mathcal{U} \tag{3.18}$$

$$x_{u,k,i} \in \{0,1\} \quad \forall (u,k) \in \mathcal{U}\times\mathcal{K} \quad \forall i \in S_k \tag{3.19}$$

$$\lambda_u^H, \lambda_u^L \in \mathbb{R}^+ \tag{3.20}$$

The decision variables are the binary variables for the placement and the continuous variables for the throughput. Prob. 3.12–3.20 is a combination of a placement and a multi-commodity flow problem. For the sake of tractability, in the next section we offer a transformation to a pure placement problem.

3.4 Pure placement problem

The aforementioned transformation is attained by fixing the continuous decision variables of the MINLP, *i.e.,* λ_u^L and λ_u^H. To do so, it is sufficient, for each application $u \in \mathcal{U}$, to fix the minimum throughput required in order to deliver the output at target rate F_u, given its configuration type and deployment region.

- *Type. 1:* processing each information unit and providing an output result should happen at rate $\frac{1}{F_u}$; by accounting for all processing and communication delay we

write

$$d_u + d_{u0} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^L}{\lambda_u^L} \leq \frac{1}{F_u} \tag{3.21}$$

which can be solved for equality in λ_u^L.

- *Type. 2:* For each application u, we have

$$d_u + d_{u0} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^H}{\lambda_u^H} \leq \frac{1}{F_u} \tag{3.22}$$

  In this case we are solving for λ_u^H; we observe that it must hold indeed $\lambda_u^H \geq \lambda_u^L$.

- *Type. 3:* if u_B is deployed in a region neighbor of the original region of u, it holds

$$d_u + d_{uj} + d_{j0} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^H}{\lambda_u^H} + \frac{\Delta_u^L}{\lambda_u^L} \leq \frac{1}{F_u} \tag{3.23}$$

  In this case, in order to have a unique solution in the minimum throughout, we impose additional constraints, namely we restrict to the set of solutions such that

$$\frac{\lambda_u^H}{\lambda_u^L} = \frac{\Delta_u^H}{\Delta_u^L} \tag{3.24}$$

Once we performed the above identification, the original problem becomes:

$$\text{maximize:} \sum_{(u,k)\in\mathcal{U}\times\mathcal{K}} f_{u,k}\, x_{u,k} \tag{3.25}$$

subject to:

$$\sum_{u\in\mathcal{U}} \mathbf{c_u}\, x_{u,k,i} \leq \mathbf{C_{k_i}}, \quad \forall k \in \mathcal{K}, \forall i \in S_k \tag{3.26}$$

$$\sum_{u\in U_k} (x_{u,k}\, \lambda_u^L + x_{u,0}\, \lambda_u^H) + \\ + \sum_{j\in\mathcal{N}(S_k)} \sum_{v\in U_j} x_{v,j} \lambda_v^L \leq B_{k0}, \quad \forall k \in \mathcal{K} \setminus \{0\} \tag{3.27}$$

$$\sum_{u\in U_k} x_{u,j}\, \lambda_u^H + \sum_{u\in U_j} x_{u,k}\, \lambda_u^H \leq B_{kj},\ \forall jk \in E, j, k \neq 0 \tag{3.28}$$

$$\sum_{k\in\mathcal{K}} x_{u,k} \leq 1 \quad \forall u \in \mathcal{U} \tag{3.29}$$

$$\sum_{k\in\mathcal{K}\setminus(\mathcal{N}(u)\cup\{u\})} x_{u,k} \leq 0 \quad \forall u \in \mathcal{U} \tag{3.30}$$

$$x_{u,k,i} \in \{0,1\}, \quad \forall (u,k) \in \mathcal{U}\times\mathcal{K}, \quad \forall i \in S_k \tag{3.31}$$

3.4.1 Complexity

Problem (3.25)–(3.31) has a formulation similar to a specific Knapsack problem, namely the Multidimensional Multiple-Choice Knapsack (MMCK) problem. This latter problem is one of the most complex versions of the Knapsack Problem's family. In this version of the KP, there are different groups of items with the constraint that exactly one item for each group must be picked [10].

In the multidimensional multiple-choice Knapsack problem there are m resource types and the amount of available resources is given by vector $C = (C^1, \ldots, C^m)$. Also, there are n disjoint classes of items, J_i $i = 1, \ldots, n$, where each class J_i has r_i items. Each item $j \in J_i$ has profit value $v_{ij} \geq 0$ and weight vector $W_{ij} = (w_{ij}^1, \ldots, w_{ij}^m) \geq 0$, where each weight component $w_{ij}^k \geq 0$, $k = 1, \ldots, m$, is the occupation of resource k of item $j \in J_i$.

The objective is to pick exactly one item from each class in order to maximize the total profit value of the pick, subject to resource constraints:

$$\text{maximize:} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{r_i} v_{ij}\, x_{ij} \tag{3.32}$$

subject to:

$$\sum_{i=1}^{n} \sum_{j=1}^{r_i} w_{ij}^k\, x_{ij} \leq C^k, \quad 1 \leq k \leq m \tag{3.33}$$

$$\sum_{j=1}^{r_i} x_{ij} = 1, \quad 1 \leq i \leq n \tag{3.34}$$

$$x_{ij} \in \{0, 1\}, \quad \forall 1 \leq i \leq n, 1 \leq j \leq r_i. \tag{3.35}$$

The pure placement problem and the MMCK have similar formulations. The next result proves NP-hardness of our problem by reduction from the MMCK problem.

Proposition 3.3. *Problem* (3.25) *is NP-hard.*

Proof. We prove the NP-hardness by reduction from a specific case of the MMCK problem. In particular, we consider the case where each class has the same number of elements r [63]. Furthermore, the NP-hardness can be easily proved by reduction from the MMCK problem with $m = 1$, since starting from $m = 1$ the problem is already known to be NP-hard (for $m = 1$ the problem is known as Multiple Choice Knapsack). Hence, for every instance of a MMCK with n classes consisting of r elements and 1 dimension, we can reduce it to an instance of our problem. Indeed, it is sufficient to consider an instance of (3.25)–(3.31) with n applications and a single region with $r - 1$ servers with infinite capacity. Each application has three possible configurations: deployed in cloud, deployed in fog or not deployed. In this manner, each application defines a class of configurations consisting of r elements. Each element's weight can be mapped to the throughput generated by each

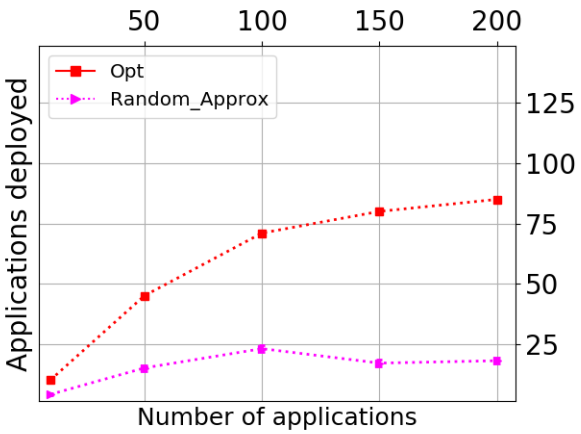

Figure 3.4: Optimal solution vs. Rounding

application towards the cloud. In this manner, the only one capacity constraint is the bandwidth constraint on the fog-cloud link. □

Several algorithmic solutions have been proposed in the literature for the MMCK problem [63, 65]. However, such heuristic solutions cannot guarantee approximation bounds on the produced allocation. Actually, to the best of the authors' knowledge, no approximation algorithm is available to date for this specific Knapsack problem with a variable number of dimensions. In [92], authors designed a polynomial time approximation scheme (PTAS) for the MMCK problem when the number of knapsack dimensions is $O(1)$. In our case, instead, the number of dimensions varies with the number of applications and regions.

One of the standard techniques to obtain approximation schemes for binary integer NP-hard problems is to relax first the integrality of choice variables thus obtaining a linear program and then round the so-obtained continuous solution of the linear program to an integer solution of the original one [72]. One technique is to consider the continuous relaxed variable as the probability to pick the tagged choice variable (setting that variable equal to 1). Typically, a difficult challenge for such a relaxation algorithm is to guarantee that the obtained integer solution is a feasible one for the initial integer problem. In our case, the rounding technique does not represent a good strategy to obtain a valid approximation scheme. As shown in Figure 3.4, the approximated feasible solution suffers a significant optimality gap with respect to the optimal one. The figure reports the best feasible solution obtained by the applying the rounding technique to the relaxed version of problem (3.25). One of the main issues is represented by the constraint (3.29), which misleads the relaxation procedure to pick solutions quite far from optimal. Indeed, by relaxing this constraint, more than one variable can assume a non-zero value. In this case, we need to choose which one to pick entirely. In our case we considered a probability

Table 3.2: Complexity of all the sub-problems

Problem	*Complexity*
Main Problem	Strongly NP-hard, no PTAS better than $n^{\frac{1}{2}-\epsilon}$
App-partitioning	Poly for $k = 2$, NP-hard for $k \geq 3$
Fog Resource Allocation	NP-hard
Pure Placement	NP-hard, No FPTAS

distribution defined for each application on the set of its possible deployments. However, as confirmed by Figure 3.4, this leads to a very conservative approximation factor since the most probable variables to be picked may be arbitrarily far from the ones chosen by the optimal integer solution.

Finally, in Table 3.2, we have resumed the complexity results for the problems tackled in this chapter. The main problem is strongly NP-hard since it induces a VNE problem; furthermore, it does not admit any tight polynomial-time approximation scheme. The application partitioning problem has a polynomial time algorithm as per the reduction showed in Proposition 3.2; it becomes NP-hard as the number of chunks is greater or equal to 3. Finally, the Fog Resource Allocation problem is formulated as a MINLP problem: by fixing the continuous throughput variables it becomes the Pure Placement problem which is NP-hard by reduction from the MMCK problem.

3.5 Fog Placement Algorithm

Hereafter, we describe FPA, a greedy solution for (3.25) which is meant to provide practically viable solution for the placement problem described above. The key intuition of the algorithm is to pick an aggregated view of fog regions' resources utilisation, thus permitting to measure the effect of placement as a pseudo-gradient descent in the space of occupied resources, while treating the alternatives for the deployment of the same application as different yet exclusive instances.

As described in Algorithm 2 FPA operates an iterative application deployment. At each step, for each region and for each application u which belongs to that region, it selects the set $\mathcal{A}$ of admissible regions for the deployment of chunk u_B. Such set includes all the regions satisfying the computational and throughput requirements of a tagged application. Preliminarily, a feasibility check is performed through a *verify* procedure: given a region and the application's requirements, it verifies whether there exists at least one server in the region to host u_B, *i.e.,* if the server has enough space in terms of CPU, memory and storage. Further, throughput requirements are verified against each configuration type for each application, by ensuring that the residual bandwidth of involved links satisfies the throughput requirement corresponding to the tagged configuration type.

The *select* procedure is reported in Algorithm 3: it first calculates, for all the admissible regions for the deployment of a given application u, a pseudo-gradient $\bar{v}_S$ ($\forall S \in \mathcal{A}$). Its

Algorithm 2: Fog Placement Algorithm (FPA)

Input: $G = (V, E), \mathcal{U}$
Output: Container placement for each $u \in \mathcal{U}$

```
1  while U ≠ ∅ do
2      place ← {};
3      for i = 1, ..., K do
4          for u ∈ U_i do
5              A ← ∅;
6              if verify(S_i, u) then
7                  A ← A ∪ {S_i};
8              for S ∈ N(S_i) do
9                  if verify(S, u) then
10                     A ← A ∪ {S};
11             if A ≠ ∅ then
12                 place[u] ← select(A,u);
13             else
14                 place[u] ← ∅;
       // select the application to be deployed
15     u* ← min_resource_region(place);
16     deploy(u*, place[u*]);
       // Update G
17     update(G, S_place[u*], S_u*, u*);
18     U ← U \ {u*}
```

components are calculated at lines 2, 3, 4, 7−9, 11, and 14, respectively, by estimating the normalized decrease of each resource type in case of deployment with tagged configuration. The output is the region with the minimum pseudo-gradient (line 16). Once a feasible region is selected for each application, Algorithm 2 chooses to be deployed first. This step is executed by *min_ resource_ region* procedure. It takes the *place* mapping as input and returns the application to which the region with the minimum pseudo-gradient has been associated by the *select* procedure.

Subsequently, once the algorithm has selected the application to be deployed, it updates the computational capacities of the server hosting the chunk of that application.

Afterwards, the algorithm updates the graph structure decreasing the available bandwidth on the links connecting the regions selected for the deployment (line 17). It iterates until all the applications have been considered.

Finally, we observe that, while the presentation of FPA is performed in the case of a batch of applications all available at the same time, the algorithmic formulation can be

Algorithm 3: *Select* procedure

Input: $\mathcal{A}$, set of admissible regions for the deployment of chunk u_B

Output: A region for the deployment

`// Build a pseudo-gradient vector for each region in` $\mathcal{A}$

for $S \in \mathcal{A}$ **do**
- $v_m \leftarrow \frac{c_u^M}{residual\_mem(S)}$;
- $v_p \leftarrow \frac{c_u^P}{residual\_proc(S)}$;
- $v_s \leftarrow \frac{c_u^S}{residual\_stor(S)}$;
- **if** $S \neq S_u$ **then**
  - **if** $S \in \mathcal{N}(S_u)$ **then**
    - `// Case 3`
    - $b_1 \leftarrow \frac{\lambda_u^H}{residual\_band(\{u,S\})}$;
    - $b_2 \leftarrow \frac{\lambda_u^L}{residual\_band(\{S,0\})}$;
    - $\bar{v}_S \leftarrow (v_m, v_p, v_s, b_1, b_2)$;
  - **else**
    - `//` $S = S_0$`, case 2`
    - $b_1 \leftarrow \frac{\lambda_u^H}{residual\_band(\{0,u\})}$;
    - $\bar{v}_S \leftarrow (v_m, v_p, v_s, b_1, 0)$;
- **else**
  - `// Case 1`
  - $b_1 \leftarrow \frac{\lambda_u^L}{residual\_band(\{0,u\})}$;
  - $\bar{v}_S \leftarrow (v_m, v_p, v_s, b_1, 0)$;

return $\arg\min_{S \in \mathcal{A}} \{\|\bar{v}_S\|^2\}$

easily adapted to the online case, where applications to be deployed arrive and depart, by simply including the release of resources to the update procedures.

3.5.1 Complexity

The computational complexity of FPA is derived by noting that procedures *verify*, *update-Server* and *update* have constant time complexity. Procedure *select* computes a vector for each eligible region in set $\mathcal{A}$. In the worst case, the cardinality of $\mathcal{A}$ is at most $K-1$. Hence, the complexity of the *select* procedure is $O(K)$. The cardinality of $\mathcal{U}$ is U, and the maximum cardinality of a neighbourhood of a certain region is $O(K)$ in the worst case. The cardinality of the set of applications to be ranked is $O(U)$ at each step. Hence, the total complexity of FPA is $O(U^2 \cdot K^2 + U^2)$ in the worst case.

Table 3.3: Distribution of the applications' microservices requirements of CPU, memory, storage and throughput.

Requirement	*Mean Value (u_0)*	*Range ($u \in \mathcal{U}$)*
CPU (c_u^P)	1250 MIPS	$[500, 2000]$ MIPS
Memory (c_u^M)	1.2 Gbytes	$[0.5, 2]$ Gbytes
Storage (c_u^S)	3.5 Gbytes	$[1, 8]$ Gbytes
Throughput ($\lambda_u(m, n)$)	3 Mbps	$[1, 5]$ Mbps

3.6 Numerical Results

In this section we evaluate the performance of the combined scheme using throughput-aware application partitioning and placement. We compare the performance with the case of throughput-agnostic application partitioning, and with the case of placement driven by virtual network embedding [114].

In order to understand the average behaviour of the proposed solutions, we examined the performance of the algorithms on a number of randomly-generated graphs. The network infrastructure is modelled with an undirected graph connecting a central cloud to a fixed number K of fog regions, where $K = 10$ in our experiments. Thus, the central cloud and fog regions form a star topology of cloud-to-fog connections, namely cloud-links. For every topology realization, links between two fog regions are added according to an Erdős-Rényi random graph model, where a link exists between two regions with probability q. Finally, each link in the resulting network is assigned with a bandwidth of 15 Mbps, both for cloud-links and fog-links.

A batch of fog applications is generated for each experiment; we considered $U = \{60, 70, 80, 90, 100\}$ for the comparison with network embedding approaches and $U = \{10, 50, 100, 150, 200\}$ for the comparison with the standard Kubernetes placement described in subsection 3.6.1. The demands of each application of the batch for CPU, storage, memory and throughput are modelled as uniform independent random variables. The mean value of such variables is dictated by the nominal value we measured on a benchmark application, that is a plate-recognition application packaged as a two-modules microservice where the fog microservice module can process the video stream either in cloud or on a fog node. The resulting distribution of the key parameters for the applications microservices are enlisted in Table 3.3; symbol u_0 refers to the nominal values we measured on a proprietary fog platform for the plate recognition app [96, 8]. Each application is generated as a DAG with a number of nodes sampled from the set $\{5, \ldots, 20\}$.

Finally, the probability that an application belongs to region $1 \leq k \leq K$ follows a truncated Pareto distribution of parameter α, *i.e.,* $\mathbb{P}\{R_u > k\} = k^{-\alpha}/\gamma$, where R_u is the random variable representing the index of the region assigned to the application u and normalization constant $\gamma = \sum_{h=1}^{K} h^{-\alpha}$.

In the proposed scenario, the servers available within each region belong to three

Table 3.4: Characteristics of the three classes of fog servers: low, medium and high.

Type	*CPU (MIPS)*	*Memory (GB)*	*Storage (GB)*
Low	5000	2	60
Medium	15000	8	80
High	44000	16	120

classes, depending on the resources they are equipped with, namely *low*, *medium* and *high* class. The computational characteristics are listed in Table 3.4 where CPU performances are described in terms of Million Instructions Per Second (MIPS) representing the processor's speed of a server. This standard measure [30] is relevant in a heterogeneous environment as fog computing, since all the servers in a specific fog region can mount different cores and different processors. Hence, given this heterogeneity, this measure represents a basic universal metric for processors' performances, used in other fog simulators [61], that is independent from the specific model of the processor. The number of servers per region is determined at each experiment sample as follows. Each region is meant to satisfy same fraction of the expected aggregated demand. More precisely, each region is equipped with aggregated resource vector $(1+\beta)\frac{U}{K}\mathbf{c_{u_0}}$. The parameter β is a slack parameter tuning the probability that resources available in a fog region are under-provisioned/overprovisioned compared to the aggregated demand. Finally, the servers' population of the tagged region is generated by allocating iteratively servers of different types at random until the region resource budget is exhausted.

Well-known Fog simulators and emulators presented in the literature [61, 80] do not support natively scenarios with multiple fog regions. Hence, we developed a Python-based simulator for the evaluation of the above algorithms. The Gurobi solver [62] has been used to solve the optimal placement problem (OPT). Experiments have been conducted on an Ubuntu Linux server with 32 core AMD Opteron(tm) 1.4GHz CPU and 64GB of memory. Each data point depicted is the result of an average over 30 instances where the network infrastructure is fixed and the application and servers distributions change. All the results are averaged with the corresponding 95% confidence interval.

3.6.1 Reference Algorithms

We compared our proposed scheme with two benchmark solutions. More precisely, we evaluate the application splitting and the deployment (FPA) algorithms. For the former one we make a comparison with a throughput-agnostic splitting algorithm. For the latter we implemented a state-of-the-art virtual network embedding algorithm presented in [114] and two variants of the Kubernetes scheduler [5].

Throughput-agnostic partitioning

For the evaluation of the partitioning algorithm we compare it with a throughput-agnostic method that, given an application DAG as input, performs a random cut of the application graph. In detail, the algorithm takes in input the application graph and perform a visit of the DAG. Once an order of the application nodes is induced by the visit, a random cut is chosen on the so-defined order without considering the total throughput on the chosen cut.

Network Embedding Algorithm

The general problem of deployment of a DAG-like fog-application can be seen as a network embedding problem, hence, we compare our solution with a standard algorithm. The general procedure [114] for the embedding algorithm consists of three main steps to be executed sequentially for each application:

1. *Topological sorting.* A topological sort is obtained by simply performing a modified visit of the application DAG. The DAG-like structure of the application ensures that there exists at least one topological order of the application nodes.

2. *Application's nodes mapping.* This procedure selects, for each node, a set of possible regions where the node can be deployed. Once this set is defined, one region is selected according to a fixed priority function. In our implementation, we select the region S_i that maximizes

$$res_{CPU}(S_i) \sum_{j \in \mathcal{N}(S_i)} res_{BW}(S_i, S_j), \tag{3.36}$$

   where $res_{BW}(S_i, S_j)$ and $res_{CPU}(S_i)$ indicate the residual bandwidth of physical link $\{S_i, S_j\}$ and the residual CPU capacity in region S_i, respectively. If any of the application's node has not eligible region, the application is not deployed.

3. *Application's links mapping.* If a mapping for all the modules of the application is found, the next step is to find a path between each couple of regions where two application nodes are mapped in the previous step. Given two regions assigned to two application's nodes, we compute all the paths between these two regions and we take the first path with enough bandwidth capacity for the application's link. If all the application's links are mapped to a set of paths of the network infrastructure, the application is finally deployed on the infrastructure.

Before applying the embedding procedure for each application, a deployment priority should be established among them. Hence, we sort the batch of applications on the basis of their total throughput demand. In this manner, the next application chosen for the deployment is the one with the minimum total throughput demand.

Remark: in the experimental section, we shall consider two variants of the VNE approach. The first is the one where embedding is performed only to regions neighbouring the original one for the target fog application u. The second one, VNE-MultiHop, assumes routing can be performed across all regions. The VNE-MultiHop variant may be an advantage by performing load balancing more efficiently across the whole deployment, it relies on a strong technological assumption in that it requires joint orchestration of network and application layers; conversely all other solutions envisioned in this chapter can be fully implemented at the application layer.

Kubernetes

Kubernetes scheduling algorithms consist of three main components: the *filtering* step where a set of servers is selected for each pod; the *ranking* which selects, on the basis of a specific priority function, the best server for the deployment among the ones previously selected; finally, the *deployment* step is dedicated to the final deployment of the pods on the servers selected on the previous step. In our case we selected *LeastRequestedPriority* as the priority function for the second step. In this manner each server node is ranked based on the fraction of the node resources that would be free after the deployment of the selected pod.

For the sake of a comparison with our approach, we adapted the Kubernetes scheduler to a multi-region scenario implementing two variants of the aforementioned procedure: the first one runs the basic Kubernetes algorithm in every single fog region; each region is hence thought as a separate cloud where a fog server is chosen to host the application chunks to be deployed. We observe that in this approach, only deployments of type 1 and type 2 are possible. The second approach is to consider the whole fog deployment as a unique region. Hence, in the filtering step, Kubernetes shall select all the servers able to host a tagged chunk across all fog regions.

3.6.2 Experimental Results

Application Partitioning

In Figure 3.5 we evaluate the effect of the fog partitioning algorithm. Figure 3.5a reports on the fraction of cloud-link usage for each application deployed on the network infrastructure both for the throughput-aware application splitting (FPA_M) and for the throughput-agnostic (FPA_R) [49]. It is clear from the figure that with the min-cut splitting the cloud-link usage is almost constant as the size of applications batch increases. Furthermore, the cloud-link usage of the throughput-aware partitioning is always less than the random cut, as expected. The cloud-link usage of the throughput-agnostic decreases as the size of application batches decreases for the applications deployment. Indeed, to increase the number of applications deployed FPA tries to save bandwidth towards the cloud as shown in [49]. Figure 3.5b represents the amount of fog-to-fog links usage for

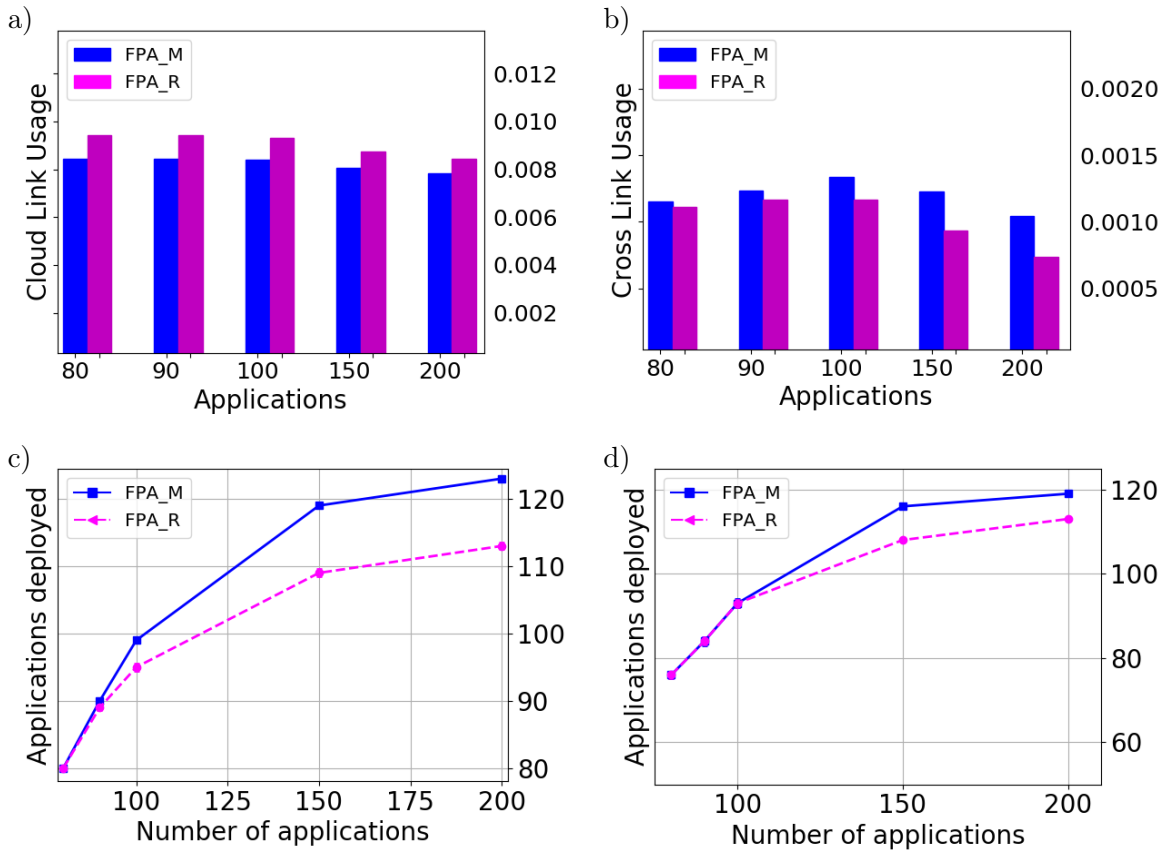

Figure 3.5: Comparison of throughput-aware vs. throughput-agnostic partitioning. a) cloud links usage, $q = 1/2$, $\beta = 0.5$; b) cross links usage, $q = 1/2$, $\beta = 0.5$; c) number of deployed applications for $q = 1/2$, $\beta = 0.5$; d) number of deployed applications for $q = 1/2$, $\beta = -0.5$.

both the two approaches. The throughput-aware partitioning exploits a larger number of cross-links: as a consequence it can deploy a larger number of applications compared with the throughput-agnostic solution, as confirmed by Figure 3.5c.

Figure 3.5c shows the effectiveness of the proposed application cut in terms of applications placement in an overprovisioning situation in terms of available computational resources. Indeed, with a minimum cut for the application splitting, FPA_M is able to deploy almost the totality of the applications' requests until $|\mathcal{U}| = 100$. This is also the reason why the cross-link usage for the throughput-aware approach keeps increasing steadily until $|\mathcal{U}| = 100$, as shown in Figure 3.5b. In fact, FPA_R suffers from the non-optimized splitting of the applications, especially in scenarios where bandwidth saturation represents a bottleneck for the applications deployment [49]. Thus, we conclude that optimized application splitting causes a significant increase of the infrastructure capacity to host fog applications. Figure 3.5d shows that the relative gain of the min-cut approach

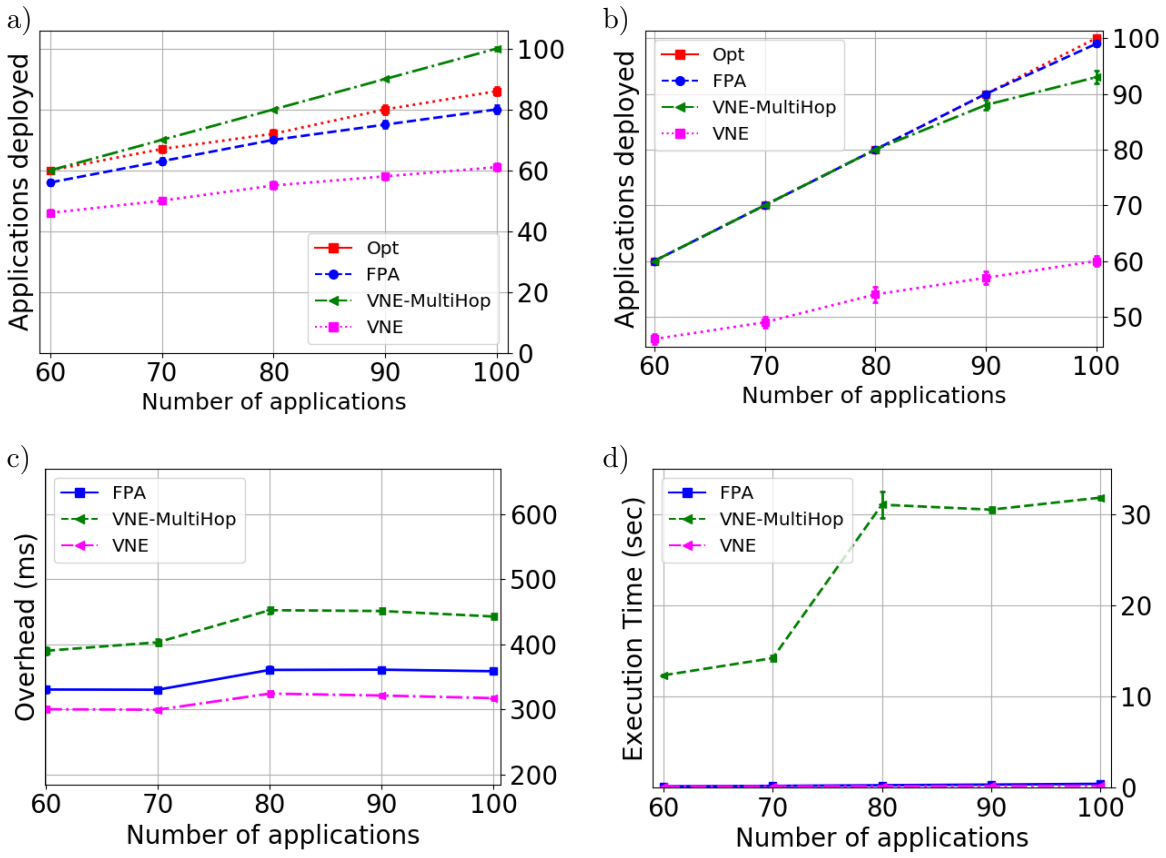

Figure 3.6: a) OPT vs. FPA vs. VNE in terms of number of applications deployed, $q = 1/2$, $\beta = 0.5$; b) OPT vs. FPA vs. VNE in terms of number of applications deployed, $q = 1/2$, $\beta = -0.9$; c) FPA vs. VNE in terms of average total delay overhead , $q = 1/2$, $\beta = -0.9$; d) Average execution time for FPA and VNE, $q = 1/2$, $\beta = -0.9$.

under tighter constraints on available resources remains significant versus throughput-agnostic approaches.

Network Embedding

For the VNE algorithm we implemented two variants: the standard one (VNE), adapted to our scenario, allows the deployment of microservices only on regions neighbouring the target one; the extended one (VNE-MultiHop), conversely, allows paths across region nodes when mapping links of the application graph during the embedding procedure.

Figure 3.6a reports on the number of applications deployed across the infrastructure in an underprovisioning scenario. We can observe that, as expected, VNE-MultiHop can deploy all the applications since we remove the constraint of using only one-hop links for the applications mappings. The difference between OPT and FPA is reduced and,

on the other hand, the VNE presents a significant loss. The VNE algorithm, indeed, tries to deploy all the applications towards the cloud until the bandwidth between the original regions and the cloud is exhausted, confirming the bandwidth towards the cloud to be a real bottleneck for the applications deployment problem. The VNE-MultiHop, instead, escape from this problem by allowing the applications' links mapping among the paths that go from the original region of the applications and the cloud. Indeed, given the metric (3.36), the best resulting region for each application deployment will be the cloud (for the unlimited computational power). However, as highlighted in Figure 3.6c, this approach can lead to a significant time overhead. In Figure 3.6b reports we have the same results when the fog has enough computational resources (overprovisioning). In this case we can see a little loss from the VNE-MultiHop still due to the metric (3.36) for the region selection. The VNE-MultiHop continues to select the cloud without considering the quantity of resources available in Fog. This can easily lead to a bandwidth saturation towards the cloud.

Figure 3.6c validates our choice of deployment using two hop schemes when offloading to neighbouring regions in Type 3 configurations. Indeed a multi-hop approach such as VNE-MultiHop may incur in a significant latency overhead due to the multi-hop path traversing several links to connect two different regions. On the other hand FPA and VNE have a small difference in delay, confirming that it is the multi-hop approach to introduce a significant latency overhead.

Finally, in 3.6d we show the execution time of the three algorithms. It is clear from the figure that VNE and FPA have comparable and scalable execution times. The VNE-multiHop presents highest execution times due to the computation of the paths between all pairs of region nodes where two application nodes are mapped.

Kubernetes

Finally, in Figure 3.7a and Figure 3.7b we have compared our solution with the two Kubernetes algorithm's variants: Kub and Kub1, respectively. In the second scenario all algorithms tend to deploy a larger number of applications than in the first one. This is expected since the latter both has more computational resources and more connected regions. In both figures we can observe that FPA performs close to the optimal solution. The poor performance of the Kub algorithm indicates that offloading towards neighbourhood fog regions is key to efficient fog resource allocation. Also, as the number of applications increases, the gap between FPA and Kub1 broadens. The reason can be ascribed to two key differences between FPA and Kub1. First, the deployment order of applications in FPA matches remaining resources at each step, by choosing the application with minimum resources consumption pseudo-gradient. In Kub1, conversely, applications are deployed in a predefined order. Second, for Kub1 neglects crosslinks bandwidth utilization, it leads quickly to bandwidth resources consumption. On the other hand, FPA's better performance is due to the fact that it accounts for bandwidth occupation of both cloud-links and

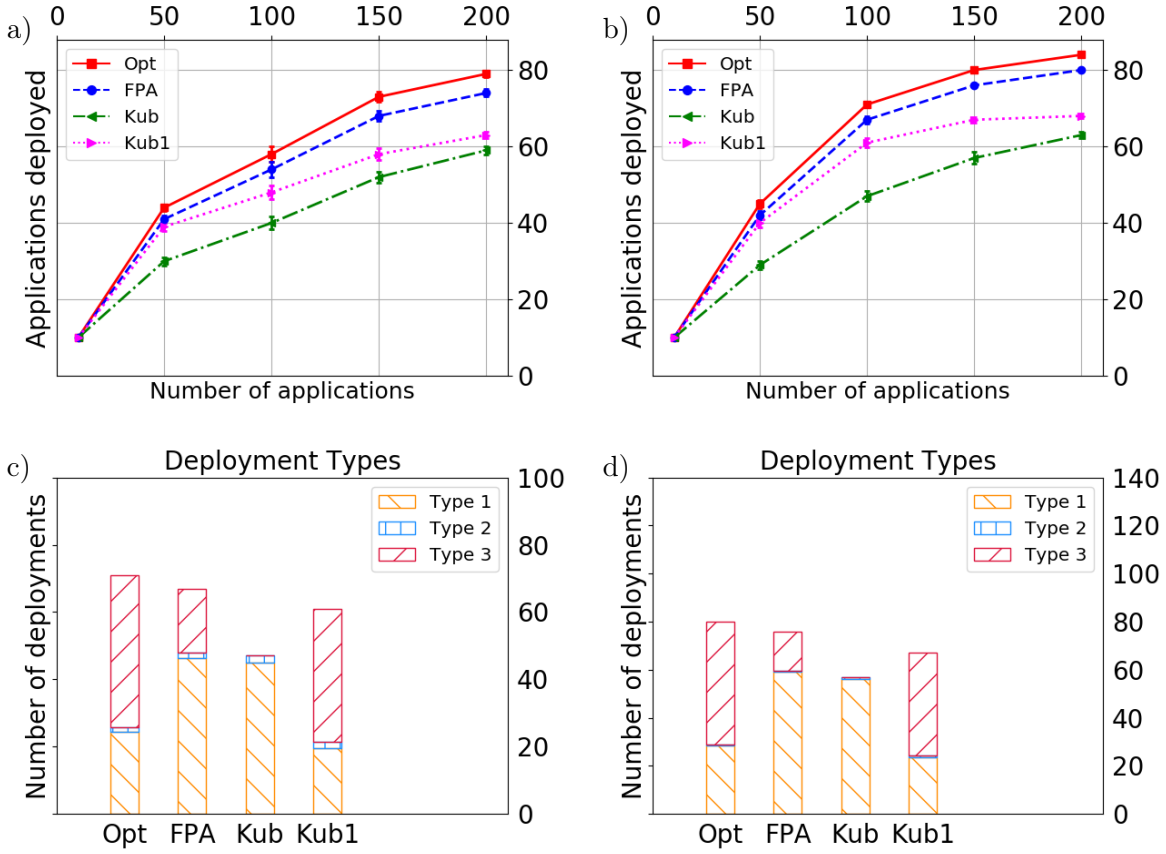

Figure 3.7: Number of deployed applications with respect to Kubernetes algorithms: a) $q = 1/3$, $\beta = -0.2$; b) $q = 0.5$, $\beta = 0.5$; c) Configuration types distribution for a typical solution instance with $U = 100$, $q = 0.5$ and $\beta = 0.5$; d) Configuration types distribution for a typical solution instance with $U = 150$, $q = 0.5$ and $\beta = 0.5$;

crosslinks. This confirms the key role of accounting for bandwidth consumption on both fog-links kind of network links for the final applications deployment, in order to avoid early bottleneck formation. Figure 3.7c and Figure 3.7d provide further insights into the structure of the produced solutions. There, we have reported the number of deployments of each type produced by different algorithms in a throughput-dominated scenario. The OPT and Kub1 solutions prioritize type 3 configurations over type 1 configurations, while the opposite occurs for FPA. Overall, as expected, deployments on fog regions, *i.e.,* type 1 and type 3 configurations, are more frequent than type 2, since they save bandwidth on cloud-links. Actually, for a batch of 150 applications the number of type 2 deployments becomes negligible (Figure 3.7d). These results confirm the importance of type 1 and type 3 deployment configurations even in an overprovisioning situation at the edge to reduce bottlenecks effects on the bandwidth towards the cloud, and, hence, maximize the number of deployed applications.

3.7 Remarks and Possible Extensions

In this chapter we have introduced a joint partitioning and optimization framework for throughput-intensive applications. We showed that a smart cut of the applications' computation flow in between cloud and fog is beneficial to cope with the applications' performance requirements while improving the revenue figures of the infrastructure owner. The scheme for the resource allocation combines a multi-commodity flow and a placement problem, but can be reduced to a specific Knapsack problem by introducing throughput proportionality and considering only the placement formulation. A greedy algorithm, FPA, is able to perform efficiently with respect to the optimal solution by placing partitioned applications using a pseudo-gradient approach. Numerical experiments confirm the scalability of the proposed fog placement scheme and the efficiency in terms of infrastructure owner's revenue and additional communication overhead compared to existing solutions in the literature.

As we previously stated, the algorithmic formulation of FPA can be easily adapted to the online case, where applications to be deployed arrive, are executed, and leave the system. Further works will involve the study of such non-clairvoyant formulation by means of the standard theoretical tools for the evaluation of online algorithms such as the competitive ratio [66] or the worst-case static regret [100].

Chapter 4

Deployment of Application Microservices in Multi-Domain Federated Fog Environments

In this chapter we consider the problem of initial resource selection for a single-domain fog provider lacking sufficient resources for the complete deployment of a batch of IoT applications. To overcome resources shortage, it is possible to lease assets from other domains across a federation of cloud-fog infrastructures to meet the requirements of those applications: the fog provider seeks to minimize the number of external resources to be rented in order to successfully deploy the applications' demands exceeding own infrastructure capacity. To this aim, we introduce a general framework for the deployment of applications across multiple domains of cloud-fog providers while guaranteeing resources locality constraints. The resource allocation problem is presented in the form of an integer linear program, and we provide a heuristic method that explores the resource assignment space in a breadth-first fashion. Extensive numerical results demonstrate the efficiency of the proposed approach in terms of deployment cost and feasibility with respect to standard approaches adopted in the literature. This chapter is mostly based on [53].

4.1 Introduction

One of the main drawbacks of existing fog platforms is that they are bound to work *within a single administrative domain.* Hence they require exclusive ownership of resources spanning from cloud to things. Nevertheless, given the wide geographical diffusion and heterogeneity of fog computing resources, single-domain solutions are likely to be unfit for the deployment of fog-native applications. In fact, they come – by their own nature – with some hard-constraining *locality* requirements, dictated by the geographical location where some processing needs to be executed or some devices need to be used.

This work contributes to current research efforts towards the definition of a multi-

domain federated fog ecosystem, where fog providers (*i.e.,* owners of fog resources) can stipulate contracts among them to rent additional resources (otherwise not accessible) and ensure smooth execution of their own applications. Whilst the technical feasibility of such approach – requiring the interface to a specific resource brokerage platform – has been proven [34, 98, 71], a different angle is considered in this work. We take the perspective of a fog provider and pose the following question: *what resources should I rent to ensure that a given set of applications is correctly deployed in the federated fog while minimizing external resource usage?*

To answer this question, we assume that the fog provider implements a mechanism (*e.g.,* based on matchmaking [77]) to retrieve from the resource brokerage platform a set of potential rentable resources owned by other providers, based, *e.g.,* on applications' locality needs. Given such a set of potential rentable resources and the self-owned resources, the objective is to find a feasible application deployment to minimize the cost of external resource rental, while deploying over the federated fog environment as many applications as possible to increase the provider's revenue.

By exploiting integer linear programming (ILP) methods, we formally formulate the *application deployment* problem: the aim is to allocate enough federated fog resources to satisfy the applications' requirements while self-owned resources are used as much as possible. More formally, an application deployment is a *map* of the applications' components, namely microservices and data flows among them, to the available fog resources. Finding such an optimal map is *virtual network embedding* (VNE) problem, which is known to be NP-hard [36, 26] and hard to approximate [14]. Hence, a practically viable approach is to design algorithmic solutions consisting of tailored heuristics to approximate an optimal solution. We thus propose a scalable heuristic algorithm solving the application deployment problem while taking into account locality constraints. Experimental results validate our solution with respect to the optimal and state-of-the-art approaches in terms of deployment cost and feasibility percentage.

4.1.1 State of the Art

4.1.2 Service deployment in federated cloud

Service deployment mechanisms have been widely studied in the literature related to cloud federation [94, 54]. Various dynamic and adaptive algorithms for resource allocation have been proposed. For instance, a distributed and adaptive approach for service placement on a heterogeneous cloud federation is presented in [35]. In [16] a multi-objective optimisation problem is formulated for resource allocation while addressing variations in applications' behaviour. Although these works describe crucial problems for an appropriate applications' deployment in a cloud federation environment, in that context locality constraints coming from inherent needs of IoT applications are not taken into account as applications' requirements. However, these constraints are the ones that mostly dis-

tinguish a federated fog from a federated cloud ecosystem, besides a higher resources heterogeneity. Conversely, our approach accounts explicitly for locality constraints when selecting resources for the deployment of fog applications. Furthermore, all the aforementioned works consider a monolithic structure for applications, whereas a microservice-oriented architecture represents a more promising paradigm for the design of future-proof fog-native applications.

4.1.3 Virtual Network Embedding

Several works in the literature considered a microservice-oriented and modelled applications as Directed Acyclic Graphs (DAGs) [116, 59]. With this kind of structure, the application deployment can be assimilated to a Virtual Network Embedding problem, which is NP-hard. Furthermore, it has been proven that VNE-like problems result hard to be approximated even with locality constraints [14]. Hence, several heuristic [32][78, 41], and metaheuristic [48, 118] methods have been proposed in the literature, especially with respect to the relevant problem of Virtual Network Function (VNF) placement [36] and for the deployment of requests of a single application (or VNF) at a time. Indeed, the most common procedure to solve these problems is to greedily deploy one VNF at a time. However, when dealing with locality constraints, the deployment should be performed considering the whole batch of applications at once. In this context, we proved that a Breadth-First Search visit driven by the applications' region-based partitioning significantly reduces the deployment costs and ensures better feasibility percentages with respect to existing solutions.

4.2 System Model

4.2.1 Involved stakeholders and scenario

Figure 4.1 resumes the scenario envisioned in this work. Two main stakeholders are involved:

- **Application provider**: it designs a *microservice-oriented application* that can be used by its customer(s) to accomplish some specific tasks. A good example is represented by the application for number-plate-based car access control described in Chapter 2.
- **Fog provider**: it owns a *fog infrastructure* that can host different microservice-oriented applications. A fog infrastructure can either be geographically distributed (spanning from cloud to edge) or confined to a specific geographical area (no cloud, only edge). It (i) provides computational, memory and storage capacity and (ii) can ensure access to things deployed on specific areas and owned by the fog provider, if any (*e.g.,* video cameras or IoT sensors). We refer to a fog infrastructure owned by a fog provider as *fog domain* (or *domain*).

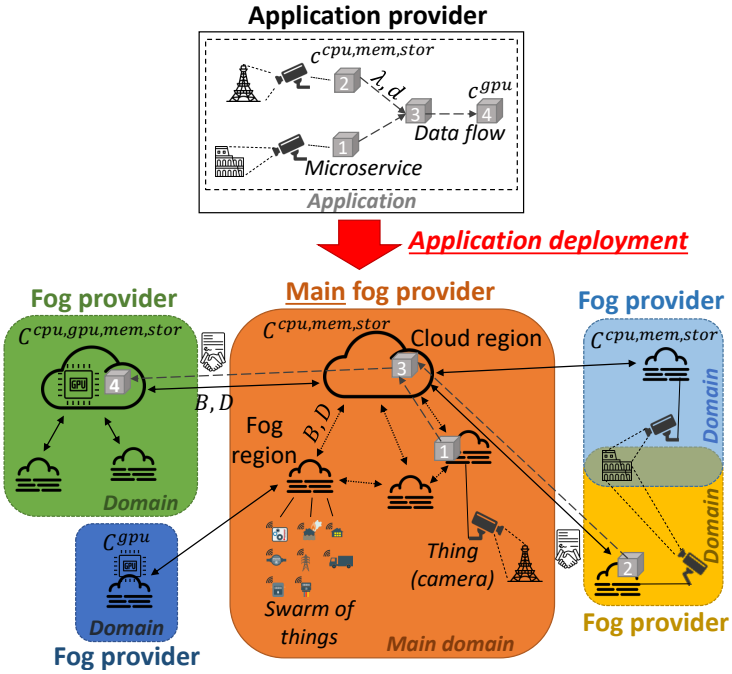

Figure 4.1: Application deployment in a multi-domain federated fog ecosystem.

As already mentioned, in our investigated scenario we take the perspective of a fog provider – called *main fog provider* from now on – that wants to deploy multiple applications over its fog infrastructure. However depending on the applications to be deployed, it may happen that the fog provider's infrastructure alone cannot meet all the applications' requirements. This may happen for two reason:

- *Locality*: an application requires some processing capability in a specific geographical location. *e.g.,* it requires video streming acquisition and elaboration from a camera owned by another fog provider.

- *Peculiar resource usage*: an application requires the usage of some peculiar resources that are not owned (or do not suffice for an appropriate application execution) by the main fog provider. For example, an application may need a GPU for fats processing, but no GPU is owned by such a provider.

From now on we assume that the main fog provider can rely on a multi-domain federated fog ecosystem (and its related infrastructure), where resources from other fog providers can be rented to meet application requirements – including locality and peculiar resource usage – otherwise exceeding its own capacity. Figure 4.1 depicts this reference scenario. In the following, the models (i) for the aforementioned multi-domain federated fog infrastructure and (ii) for the applications to be deployed are presented.

Table 4.1: Main notation used throughout the chapter.

Symbol	*Meaning*
$G_I = (V_I, E_I)$	Infrastructure network graph: V_I physical nodes (regions), E_I physical edges (network connections)
$\mathcal{A}$	Set of applications to be deployed on G_I
$G_A = (V_A, E_A)$	Graph for application $A \in \mathcal{A}$: V_A virtual nodes (modules) and E_A are virtual edges (data flows)
$L_A \subseteq V_A$	Subset of virtual nodes to be deployed on a specific physical node of the infrastructure
$r_A : L_A \to V_I$	Maps virtual node v_A to a given physical node
$c^r_{v_A}$	Resource requirements of virtual node $v_A \in V_A$, with $r \in \{cpu, gpu, mem, stor\}$
$\lambda_A(u_A, v_A)$	Max. throughput on edge $(u_A, v_A) \in E_A$
$d_A(u_A, v_A)$	Max. tolerated delay on edge $(u_A, v_A) \in E_A$
S_v	Set of available hosts in physical node $v \in V_I$
$C^r_{v,i}$	Resource capacity of the i-th host in physical node $v \in V_I$, with $r \in \{cpu, gpu, mem, stor\}$
$D_{u,v}$	Latency on the physical link $(u, v) \in E_I$
$B_{u,v}$	Capacity of the physical link $(u, v) \in E_I$
$w(v)$	Cost of physical node $v \in V_I$
P	Set of computed paths p between any couple of physical nodes
$P_{u,v} \subseteq P$	Set of computed paths between $v \in V_I$ and $u \in V_I$
s_p, d_p	First and last node of path $p \in P$

4.2.2 Multi-domain federated fog infrastructure modelling

The considered infrastructure consists of several geographically distributed fog or cloud *regions* belonging to different fog domains, including the main fog domain. From a modelling point of view, the only difference between cloud and fog regions is their processing, memory and storage capabilities. Each region is composed by multiple *hosts* (or servers), which are the atomic unit where a microservice can be deployed. An example of such an infrastructure is shown in Figure 4.1.

The multi-domain infrastructure is described by a weighted graph $G_I = (V_I, E_I)$ where V_I is the set of regions of the infrastructure and $E_I \subseteq V_I \times V_I$. Furthermore, cost function $w : V_I \to \mathbb{R}^+$ specifies the cost for application deployment in a specific region of the infrastructure. Nodes (regions) and links/edges composing the infrastructure are also called *physical nodes* and *physical links/edges*. Each physical link $(u, v) \in E_I$ is characterized by a couple $(D_{u,v}, B_{u,v})$ modelling the latency and the bandwidth capacity of the link, respectively. We assume that hosts within the same region are interconnected through high-performance connections, where bandwidth is never a bottleneck and latency can be considered negligible. Conversely, regions (either belonging to different domains or within the same domain of the federation) are interconnected through best-effort network connections, meaning that bandwidth and latency constraints must be considered.

4.2.3 Application modelling

We denote $\mathcal{A}$ as the set of applications to be deployed on the infrastructure and each application $A \in \mathcal{A}$ is described by a weighted DAG $G_A = (V_A, E_A)$, where V_A is the set of microservices (or modules) composing the application, and E_A represents the set of dependencies between microservices. Each directed edge (u_A, v_A) of an application A is characterized by (i) the maximum throughput generated on that link, $\lambda_A(u_A, v_A)$ and (ii) the maximum tolerated delay on that link, $d_A(u_A, v_A)$. Each node v_A of an application A has computational requirements in terms of CPU, memory and storage $c^r_{v_A}$, where $r \in \{cpu, mem, stor\}$. Furthermore, each application has a set of *locality constraints* representing the regions where some microservices must be deployed, for example, to acquire data from specific devices belonging to that regions. We model this fact by introducing a set $L_A \subseteq V_A$ for each application $A \in \mathcal{A}$. This set contains all the nodes that need to acquire data from a tagged device placed on a specific region: hence, they must be placed in that region. Function r_A specifies, for each node in L_A, the region where it must be deployed. Nodes and links composing an application are also called *virtual nodes* and *virtual links*, respectively.

4.3 Problem Formulation

In this section we provide the ILP formulation of the placement problem.

4.3.1 Decision variables

We introduce two types of variables:

1. A binary variable for the assignment of each application module to a physical node:

$$x_{v,i}^{A,v_A} = \begin{cases} 1, & \text{if module } v_A \text{ of application } A \\ & \text{is deployed on host } i \text{ of node } v \\ 0, & \text{otherwise} \end{cases}$$

   where $A \in \mathcal{A}$, $v_A \in V_A$, $v \in V_I$ and $i \in S_v$.

2. A binary variable for the assignment of virtual links of each application to physical paths:

$$y_p^{(u_A,v_A)} = \begin{cases} 1, & \text{if link } (u_A, v_A) \text{ is mapped to path } p \\ 0, & \text{otherwise} \end{cases}$$

   where $A \in \mathcal{A}$, $(u_A, v_A) \in E_A$, and $p \in P$.

4.3.2 Objective Function

Given a subset of nodes belonging to the main provider, we want to minimize the total *deployment cost* for all the application requests of that provider. We assign a weight w to the deployment of a virtual node onto a physical node v of the federated infrastructure. We assume that weights are larger when such deployment is performed towards nodes belonging to other fog domains, since their resources need to be rented. More formally, we have a weight function defined for each physical node, $w : V_I \to \mathbb{R}^+$. Finally, the objective function writes as follows

$$\sum_{A \in \mathcal{A}} \sum_{v_A \in V_A} \sum_{v \in V_I} \sum_{i \in S_v} w(v)\, x_{v,i}^{A,v_A}. \tag{4.1}$$

4.3.3 Constraints

First, we have integrity constraints on the applications' deployment on the infrastructure. Indeed, all the modules of an application must be deployed and each such a module must be placed only once. These conditions are expressed through the following constraint (4.2) and (4.3), respectively.

$$\sum_{v_A \in V_A} \sum_{v \in V_I} \sum_{i \in S_v} x_{v,i}^{A,v_A} = |V_A|, \quad \forall A \in \mathcal{A}, \tag{4.2}$$

$$\sum_{v \in V_I} \sum_{i \in S_v} x_{v,i}^{A,v_A} = 1, \quad \forall A \in \mathcal{A}, \forall v_A \in V_A. \tag{4.3}$$

Second, we have constraints on the resource capacity for each host in the infrastructure:

$$\sum_{A \in \mathcal{A}} \sum_{v_A \in V_A} c^{res}_{v_A} x^{A,v_A}_{v,i} \leq C^r_{v,i} \tag{4.4}$$

where $v \in V_I$, $i \in S_v$, and resource $r \in \{\text{cpu, mem, stor}\}$.

Third, there are constraints related to virtual and physical links capacity. We start by introducing the structural constraints binding variables related to nodes and virtual links:

$$\sum_{p \in P_{u,v}} y^{(u_A,v_A)}_p \leq \sum_{i \in S_u} x^{A,u_A}_{u,i}, \tag{4.5}$$

$$\sum_{p \in P_{u,v}} y^{(u_A,v_A)}_p \leq \sum_{j \in S_v} x^{A,v_A}_{v,j}, \tag{4.6}$$

$$\sum_{p \in P_{u,v}} y^{(u_A,v_A)}_p \geq \sum_{i \in S_u} x^{A,u_A}_{u,i} + \sum_{j \in S_v} x^{A,v_A}_{v,j} - 1, \tag{4.7}$$

where $(u, v) \in V_I \times V_I$, $A \in \mathcal{A}$, and virtual link $(u_A, v_A) \in E_A$.

Constraints (4.5), (4.6) and (4.7) ensure that a unique physical path is used by a virtual link whenever the virtual nodes connected by such link are the extreme nodes of the path.

The following are constraints on the bandwidth capacity for all physical links $(u, v) \in E_I$

$$\sum_{A \in \mathcal{A}} \sum_{(u_A,v_A) \in E_A} \sum_{p \in P:(u,v) \in p} \lambda_A(u_A, v_A)\, y^{(u_A,v_A)}_p \leq B_{u,v}, \tag{4.8}$$

and the applications' delay constraints, namely

$$y^{(u_A,v_A)}_p \sum_{(u,v) \in p} D_{u,v} \leq d_A(u_A, v_A), \tag{4.9}$$

where $A \in \mathcal{A}$, $(u_A, v_A) \in E_A$, and $p \in P$.

Finally, we have locality constraints: they impose the placement of a subset of applications' nodes on specific regions, since data are acquired by devices located on those specific regions:

$$\sum_{i \in S_{r_A(v_A)}} x^{A,v_A}_{r_A(v_A),i} = 1, \quad \forall A \in \mathcal{A}, \forall v_A \in L_A. \tag{4.10}$$

4.4 Proposed Solutions

The problem formulated in the previous section is a VNE problem which, as already said, is a well-known NP-hard problem. Hence, it is important to look for fast and scalable algorithms whose output solutions have an acceptable cost.

The VNE problem has been extensively studied in the literature, and several heuristic solution methods have been proposed [32]. In the following, we briefly resume the main idea of the most adopted greedy approaches, which are based on a Depth-First Search (DFS) exploration of the deployment solution space. We start from this approach and propose a solution tailored for the specific requirements of our problem, based instead on a Breadth-First Search (BFS) method.

4.4.1 Depth-First Search Approach

The most popular heuristic methods for VNE adopt a greedy embedding procedure. It receives as input a sorted list of application deployment requests and returns a mapping between each request and a subset of physical nodes/edges of the infrastructure. Our reference DFS-based algorithm is an adaptation of the basic one presented in [114].

For each application in the input batch, the following three steps are executed sequentially:

1. *Topological sorting of the application graph*: it is obtained by performing a visit of the application graph; as a result, an order of the DAG nodes is established.

2. *Virtual node mapping*: it defines, for each virtual node, a set of possible placements (set of possible regions where the virtual node can be placed). Once defined, it selects one placement from this set.

3. *Virtual link mapping*: it finds a path between each couple of physical nodes where virtual nodes are mapped in the previous step.

The DAG-based deployment order on the application's modules, as established in the first step, accounts for their dependencies. A topological sort, indeed, ensures that for each couple of nodes $u_A, v_A \in V_A$, if $(u_A, v_A) \in E_A$ then u_A precedes v_A in the topological order, that is u_A is a *predecessor* of v_A. Once the order is defined by the topological sort, the application's node mapping to the infrastructure's resources is performed. At this stage, node mapping accounts for the actual resources occupation, and requires to establish a certain priority function; it defines, for each application, an admissible set for each module of the applications containing all the regions with at least one host with enough resources for that module. If this set contains at least one region, it selects the region that $v \in V_I$ that maximizes

$$res_{CPU}(v) \left\{ \sum_{u|(u,v)\in E_I} res_{BW}(u,v) + \sum_{u|(v,u)\in E_I} res_{BW}(v,u) \right\}, \tag{4.11}$$

where $res_{BW}(u,v)$ and $res_{CPU}(v)$ are the residual bandwidth of the physical link (u,v) and the total residual CPU capacity in region v, respectively [114]. If the set of admissible regions is empty for at least one module of the application, such an application cannot be

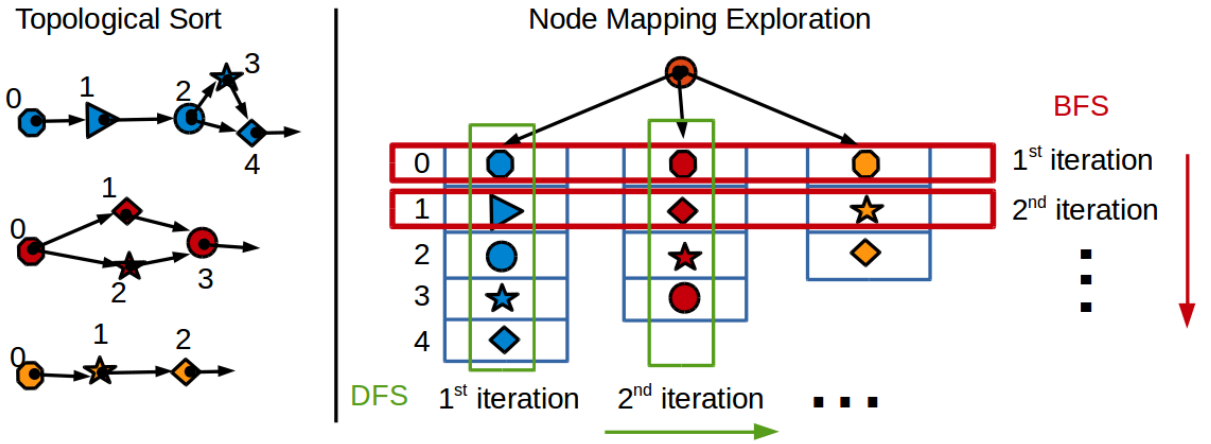

Figure 4.2: DFS vs. BFS approaches after the topological sorting step.

deployed. Instead, if a mapping for all the modules of the application is found, the next step is the link mapping procedure. For each virtual edge, the algorithm takes the two physical nodes assigned to the virtual nodes it connects (chosen by the second step) and considers the least-congested path that satisfies both bandwidth and delay requirements of the virtual link. If all the virtual links can be mapped to a non-empty set of paths of the infrastructure, the algorithm has found a complete map of the application onto the physical infrastructure. This operation is iterated for all the applications to be deployed in the input sorted list.

4.4.2 Breadth-First Search Approach

The basic idea of our novel approach is to deploy the batch of applications in a breadth-first fashion. This means that we do not deploy a single application per time, conversely, at each step we consider all the application nodes: at each iteration we consider for placement the first virtual node of each application, as determined by the topological sorting of the application's graph. Every time a virtual node is mapped to a host, it is popped out from the stack of the topological order of its application. The rationale of this approach is that applications' locality constraints can be better matched, especially the one requiring a placement in the main domain (*e.g.,* for privacy reasons).Indeed, a depth-first greedy procedure can quickly saturate all the resources of a particular region for the deployment of certain applications without considering that some remaining (not deployed yet) applications in the batch may have hard locality constraints on that region. This typically renders the deployment of the whole batch infeasible,as will be better highlighted in the next section.

Given a batch of applications to be deployed on the multi-domain infrastructure, our algorithm consists of three main steps:

1. *Sorting of the batch of applications*: it sorts all the application on the basis of their

total bandwidth consumption.

2. *Topological sorting of the application graph*: it establishes a topological sort of each application graph following the applications' order decided in the previous step. A stack is created for each application: the first node of the topological order is put on top of the stack.

3. *Virtual node and link mapping*: it iterates over all the applications in a breadth-first manner, exploring all of them jointly and level by level. At each iteration, all the virtual nodes on top of the applications' stacks are popped up and a node mapping is performed in the order established by step 2: each module is assigned to a host of the infrastructure. Link mapping is then performed in conjunction with node mapping. Figure 4.2 illustrates the iterations of the third step of the algorithm (after the topological sort of the applications' DAGs) comparing this breadth-first approach with respect to a depth-first visit.

More in detail, the selection of the regions to perform the virtual node mapping (step 3) is operated by choosing for deployment a set of admissible regions that takes into account the requirements and the locality constraints of each application module. Once such set is defined, the algorithm selects the regions with the lowest *deployment cost*. Afterwards, to choose the most suitable region, it greedily selects the placement option that has the smaller relative increment of occupied nodes' resources with respect to CPU, memory, storage and bandwidth.

Finally, to perform link mapping, each time an application node v_A is mapped to a region r, the algorithm takes the list of all the regions assigned to the predecessors (in the topological order of A) of v_A and selects the least-congested paths between them and r.

Figure 4.3 illustrates a toy example to show the difference between the two approaches in the deployment phase of applications. In this example, we have two applications composed by three modules and an infrastructure with two domains, the Domain A with deployment cost equal to zero, and the Domain B with a deployment cost greater than zero. Both the domains have three hosts with the same capacity and only one microservice can be deployed on each host. The first modules of each application have the locality constraint that requires to deploy that modules on one server of the first domain. A DFS approach will firstly deploy all the microservices of the first application in the domain with lower cost, and then it will take into account the second application. However, given its greediness with respect to the cost function, the DFS will saturate Domain A with only modules from the first application leaving no space for the second application in the same domain. This leads to an infeasible solution for the DFS approach since it is not able to satisfy the locality constraint for the second application. The BFS approach, on the other hand, at each iteration will consider both the two applications, deploying, at iteration 0, the first modules of applications on the first domain, satisfying their locality constraints. The approach will continue to deploy the other modules using also resources from Domain

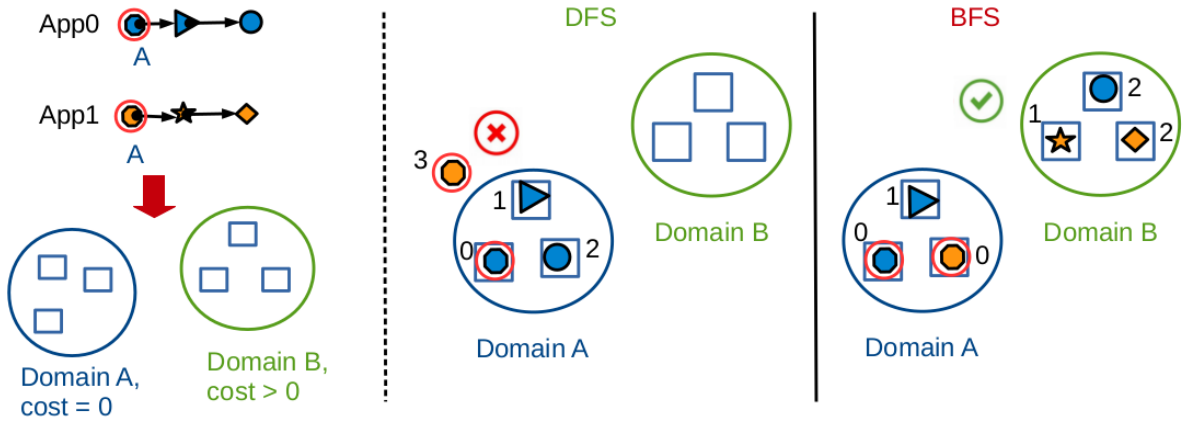

Figure 4.3: Example of applications deployments performed by the DFS and the BFS.

Table 4.2: Applications' microservices requirements [26].

Requirement	*Mean Value*	*Range*
CPU ($\mathbf{c_{v_A}}$cpu)	1250 MIPS	$[500, 2000]$ MIPS
Memory ($\mathbf{c_{v_A}}$mem)	1.2 Gbytes	$[0.5, 2]$ Gbytes
Storage ($\mathbf{c_{v_A}}$stor)	3.5 Gbytes	$[1, 8]$ Gbytes
Throughput (λ_A)	3 Mbps	$[1, 5]$ Mbps
Delay (d_A)	262.5 ms	$[25, 500]$ ms

B. In this manner all the applications will be deployed without any constraint violation.

4.5 Performance Evaluation

In this section we validate our solution for applications deployment on a multi-domain federated infrastructure. Our goal is twofold: (i) prove that a Depth-First Search approach for the deployment of applications can negatively impact on either the *feasibility* of the solution (*i.e.,* not all the applications can be deployed) or on its *optimality*; (ii) show that a Breadth-First Search approach leads to a good trade-off between feasibility and optimality, especially when some locality constraints are specified. We measure the *feasibility* as the percentage of instances that admit a feasible solution (*i.e.,* meets all the constraints), for the problem described in Section 4.3, among all the generated instances.

4.5.1 Simulation settings

We describe how we generate the test network topologies and the batch of applications to be deployed. *Network topology:* the multi-domain fog infrastructure is modelled as a directed network graph with a number of fog regions K and a number of domains D. For

the main fog domain, we consider a central cloud region that is always connected to fog regions in a star topology. For each randomised topology realisation, links among different fog regions (either belonging to the main fog provider or to other external providers) are instead added according to an Erdős-Rényi random graph model, where a link between two regions exists with probability $pr = 0.5$. Eventually, each link in the resulting topology is assigned an average bandwidth of 60 Mbps and an average delay of 10 ms representing the average values of modern communication links [26]. The hosts available within each region belong to three classes, depending on the resources they are equipped with, namely *low* (CPU: 5000 MIPS, memory: 2 GB, storage: 60 GB), *medium* (CPU: 15000 MIPS, memory: 8 GB, storage: 80 GB) and *high* (CPU: 44000 MIPS, memory: 16 GB, storage: 120 GB). To well dimension the computational resources within any region, the aggregated demand of the batch of applications to be deployed (in terms of CPU, memory and storage) is equally split among all the regions excluding the main cloud. Then, the set of hosts of a certain region is generated by iteratively and randomly choosing hosts of different types until the aggregated demand fraction assigned to that region is satisfied.

Application batch: a batch of applications $\mathcal{A}$ is generated for each experiment; we consider $|\mathcal{A}| = \{10, 15, 20, 25, 30\}$. The demand of each applications' module in terms of CPU, storage, memory and throughput are uniform independent random variables. The distribution values for each microservice are enlisted in Table 4.2. Each application is generated as a DAG ordering all the nodes and adding an edge only between predecessors and successors in the order.

We developed a Python-based simulator for the evaluation of the above algorithms since well-known fog simulators do not support scenarios with multiple domains and fog regions yet [61]. The Gurobi solver [62] has been used to solve the optimal placement ILP problem (OPT). Each data point in the shown graphs is the average value over 30 randomized instances, where the infrastructure is fixed and the batch of applications and host distribution change. All the results are shown with their corresponding 95% confidence interval. We evaluate the proposed solutions in a scenario with $K = 6$ regions, $D = 3$ domains. The optimization problem is solved from the *main* fog provider perspective: such domain contains one cloud and one fog region, while the other fog regions are distributed among the remaining fog providers' domains (*external* providers). We consider a unit cost for all the deployments outside the main domain ($w = 1$) and zero-cost for the deployments inside the main domain ($w = 0$). To highlight the importance of *locality constraints*, we impose that the first module of each application must reside in the fog region of the main domain.

4.5.2 Numerical results

In Figure 4.4 we evaluate the tradeoff between optimality and feasibility of the proposed solutions. Figures 4.4a) and b) report on feasibility and optimality with two variants of the state-of-the-art DFS approach. *DFS_SoA_NoCost* represents the variant where

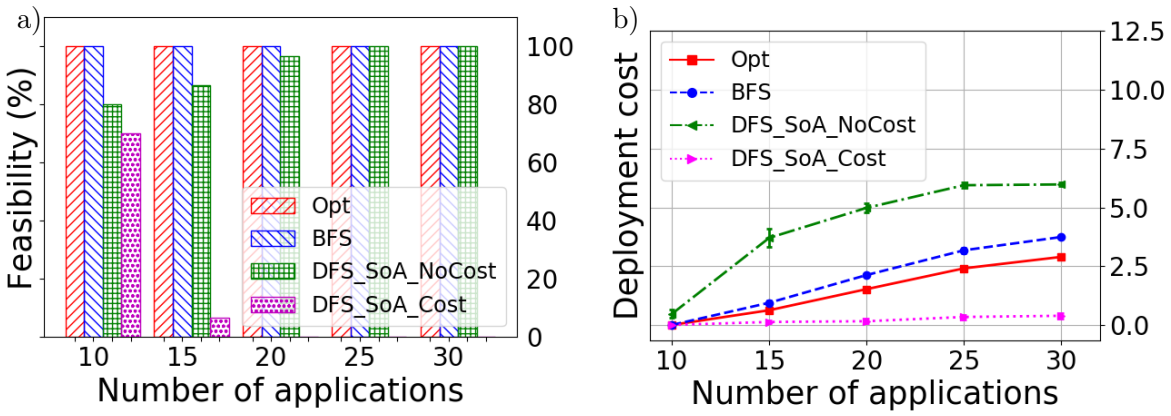

Figure 4.4: Feasibility-optimality tradeoff. a) Feasibility percentage; b) Total deployment cost for each application.

deployment cost optimization is not taken into account: this means that the objective of this strategy is just maximizing the number of deployed applications without caring about the overall deployment cost. Conversely, *DFS_SoA_Cost* objective is to minimize the deployment cost. *BFS* refers instead to our proposed novel strategy. From Fig. 4.4a), we can notice that *OPT*, *BFS* and *DFS_SoA_NoCost* have high feasibility. Especially, the first two strategies have a feasibility of 100%, meaning that they are able to deploy the complete batch of applications in all the 30 randomized instances (note that *feasibility* refers to the percentage of instances that are feasible).

Additionally, looking at the deployment cost (computed as in eq. 4.1) in Fig. 4.4b), we can see that *BFS* presents a very close solution to the optimal one (*OPT*). On the other hand, *DFS_SoA_NoCost* leads to a high deployment cost even though it has a good feasibility percentage. Conversely, in *DFS_SoA_Cost* the feasibility percentage is low as well as the deployment cost. This behaviour is reasonable given the greedy nature of the DFS approach. Indeed, if we include a cost optimization in such an approach, the algorithm prioritizes all the regions with the lowest cost (that is, the regions in the main domain) for the deployment of all the applications' modules. In this manner, resources with lower cost are quickly saturated precluding the possibility to satisfy the locality constraints for the applications that have not been deployed yet. On the other side, if we do not consider cost optimization, all the regions are treated in the same way, increasing the chance of having a feasible solution while increasing the deployment cost too. From this perspective, a BFS approach is beneficial since it does not evaluate the deployment of each application at a time, but it considers the deployment of a part of every application at each iteration. Thanks to this property, this method leads to a high feasibility percentage, since it helps to guarantee locality constraints, and to a strong reduction of the deployment cost. In summary, Fig. 4.4b) confirms that our solution explores the best tradeoff between

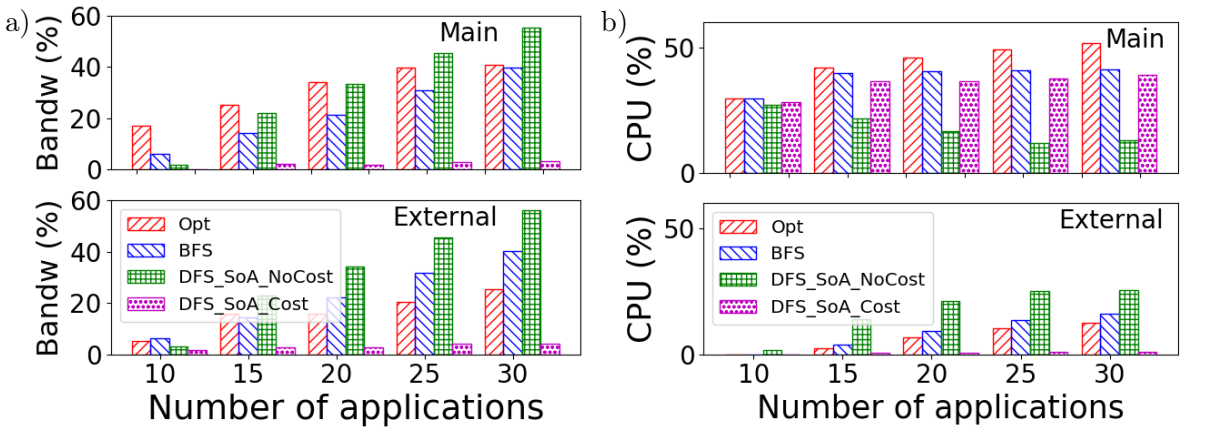

Figure 4.5: Bandwidth and CPU usage. a) Percentage of bandwidth usage within the main domain and in external domains; b) CPU usage in the main domain and in external domains.

optimality and feasibility with respect to both variants of DFS.

Figure 4.5a) reports on the bandwidth usage of the proposed solutions within and outside the main domain. The bandwidth consumption of *DFS_SoA_Cost* is almost constant and low in both the main and external domains since it tries to avoid the deployment of applications towards external domains and mostly deploy applications on the main fog region until it becomes saturated. *OPT* and *BFS* solutions present, instead, a similar trend on bandwidth usage as the size of the application batch increases. The consumption of the link between the main cloud and fog region is slightly higher for *OPT* than for *BFS* since the optimal deployment can accommodate more applications in the main domain. With respect to external domains, the two solutions behave the opposite confirming the slightly higher cost of the *BFS* solution. Finally, *DFS_SoA_NoCost* presents a similar behaviour on both main and external domains, since it maximizes relation (4.11) without distinguishing between main/external domains.

In Figure 4.5b) we report the percentage of CPU usage of all the proposed solutions on the main domain (upper figure) and on external domains. Reasonably, the CPU usage of *OPT* is slightly greater than of *BFS* in the main domain since it deploys more applications there, as confirmed by Figure 4.4b). *DFS_SoA_Cost*, given its greedy nature, presents a high and constant percentage of CPU usage in the main domain, meanwhile it has very low CPU consumption in external domains. On the other hand, the *DFS_SoA_NoCost* has the opposite trend, showing an increasing usage from external domains and decreasing usage from the main domain as the number of applications increases. Note that memory and storage usage have a similar trend as CPU usage and thus are not reported in this section for the sake of conciseness.

Finally, Table 4.3 reports on the average execution time of the proposed solutions over the 30 instances. The values of *OPT* refer to executions that are stopped after

Table 4.3: Execution time (sec).

$\|\mathcal{A}\|$	*OPT*	*BFS*	*DFS_SoA_NoCost*	*DFS_SoA_Cost*
10	3.58	0.03	0.02	0.02
15	26.90	0.05	0.05	0.03
20	40.30	0.07	0.07	0.03
25	60.72	0.09	0.09	0.03
30	79.10	0.13	0.12	0.04

5 minutes if the solver has not completed the computation in that time range. The higher scalability of all heuristic approaches is evident compared to *OPT*. Note that *DFS_SoA_Cost* has lower execution time than the other approaches because its execution is generally stopped earlier,*i.e.,* when the algorithm cannot deploy one of the applications and the deployment is considered infeasible. The time efficiency of heuristic methods is given by their polynomial time complexity: indeed, we can see the deployment of the batch of applications as a visit of a graph composed by a root dummy node connected with an edge to all the subgraphs represented by the applications, as shown in Figure 4.2. In this problem, given the existence of locality constraints, a breadth-first visit results to be more efficient in terms of feasibility and optimality.

4.6 Remarks and Possible Extensions

In this chapter we have considered the problem of deploying fog applications onto a federated cloud-fog environment. In this context, solving the problem of initial resource selection is crucial to reduce offloading costs, satisfy all the applications' requirements and accommodate future requests. By considering a microservice paradigm for fog applications, a virtual network embedding problem is faced, which is known to be NP-hard. In this context, standard heuristic solutions need to trade-off feasibility, cost-efficiency and scalability. This work proposed a new deployment technique for batches of fog applications, based on a breadth-first visit of all the applications' graphs to deal with locality constraints in the deployment. It has been showed to provide a near-optimal performance and yet good percentage of feasibility, outperforming standard depth-first greedy heuristics.

Future works will investigate the applications deployment in case of several locality constraints for each application. Indeed, one limitation of the present solution is that it has been tested only with a single locality constraint for each application. The BFS algorithm can be extended in case of different locality constraints defined for each application. The idea is to initially partitioning all the applications based on their locality constraints and then perform the placement exploring the solution space in a breadth-first fashion. In fact, from numerical results, one of the most frequent cause of infeasibility is represented by the impossibility to satisfy locality constraints due to initial greedy choices that compromise

the complete deployment of the batch of applications. Prioritizing locality constraints can reduce the probability of violating them in overprovisioned situations, *i.e.,* with enough computational and networking resources.

Furthermore, in future works, we shall extend the proposed framework to account for transactions between different domains, paving the way to the design of new exchange mechanisms for fog computing.

Chapter 5

Optimal Blind and Adaptive Fog Orchestration under Local Processor Sharing

In this chapter we study the tradeoff between running cost and processing delay in order to optimally orchestrate multiple fog applications. Fog applications process batches of objects' data along chains of containerised microservice modules, which can run either for free on a local fog server or run in cloud at a cost. Processor sharing techniques, in turn, affect the applications' processing delay on a local edge server depending on the number of application modules running on the same server. The fog orchestrator copes with local server congestion by offloading part of computation to the cloud trading off processing delay for a finite budget. Such problem can be described in a convex optimization framework valid for a large class of processor sharing techniques. The optimal solution is in threshold form and depends solely on the order induced by the marginal delays of N fog applications. This reduces the original multidimensional problem to an unidimensional one which can be solved in $O(N^2)$ by a parallelised search algorithm under complete system information. Finally, an online learning procedure based on a primal-dual stochastic approximation algorithm is designed in order to drive optimal reconfiguration decisions in the dark, by requiring only the unbiased estimation of the marginal delays. Extensive numerical results characterise the structure of the optimal solution, the system performance and the advantage attained with respect to baseline algorithmic solutions. This chapter is mostly based on [44].

5.1 Introduction

In a typical fog or cloud service, data batches generated from objects are processed sequentially over a sequence of containerised microservice modules [15, 115]. In principle, the corresponding virtual machines or containers can be run either in the central cloud

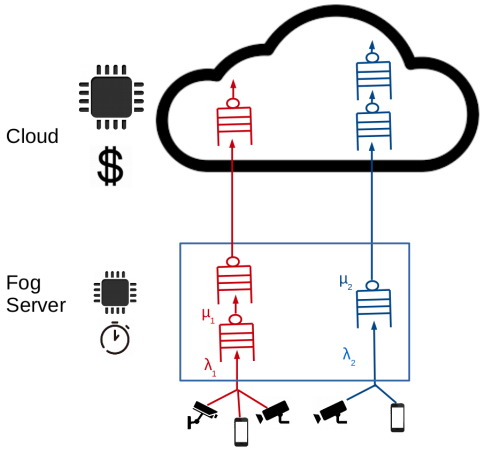

Figure 5.1: Illustration of the cost vs processing tradeoff in the considered fog placement problem.

or hosted on a fog server. In this context, the fog orchestrator is the controller which decides for each application which part of the computation, *i.e.,* which modules, should run on the fog server and which ones in cloud [87, 115]. The resulting placement has a key impact on the processing delay experienced by fog applications. In fact, even a top-notch edge server has limited capacity compared to the aggregated capacity of overprovisioned cloud systems, so that processor sharing on a fog server may induce unacceptably long processing delays.

In fact, optimization of computation capacity of edge units to meet customers' demands is emerging as a central problem in fog computing [75, 117]. In this work, fog applications are assumed to receive data batches to process at given rate. Each data batch entails a given processing delay when processed by an application module.

Under processor sharing, concurrent applications' modules run in parallel on the same fog server, at the price of increased processing delays as shown in Figure 5.1. However, in case of performance degradation, it is still possible to migrate part of the running applications to the cloud. Throughout this work, the objective is to study how to tradeoff between the load on local edge servers, reflecting in the processing delay, and the cost for offloading to the cloud. To this aim, the fog orchestrator can tune the performance of the system by increasing or decreasing the number of modules of an application which are executed on the fog server. The resulting control problem is a convex problem where the control is the number of modules to be run in fog for each application.

5.1.1 Main Contributions

The first part of this work focuses on the structure of the solution by minimising the cumulative batch processing delay. Under a general processor sharing policy, the optimal solution is determined by 1) the budget expenditure and 2) by a crucial metric, namely the *marginal delay* of applications. The marginal delay represents the performance gain

obtained by offloading a module of a tagged application towards the cloud. The N-dimensional optimization problem can thus be solved via a one-dimensional search in the space of the spent budget. The optimal solution sorts applications in order of increasing marginal delays. The optimal policy is of threshold type in all cases of practical interest, randomised on at most one control.

In the second part of the work, the precise knowledge of the optimal solution suggests a polynomial time algorithm to determine the optimal placement in $O(N^2)$ time. Furthermore, the optimal policy can be adjusted in case the system load varies in time using an adaptive algorithm based on stochastic approximations of the Robinson-Monroe type, whose convergence proof is derived for a primal-dual convex minimisation using the ODE method [74]. It is worth noting that this type of solution is able to converge to the optimal threshold policy in the dark: this is key when there is no apriori information on some system parameters. Specifically, this is crucial when facing unknown data batch arrival rates or unknown applications' processing rates, either in cloud, in fog or both. More in general, the proposed algorithm converges to the optimal solution leveraging only on online unbiased estimates of the processing delay. More important, provided that processing delays are increasing and convex in the server's load, blind convergence to the optimal restpoint shall hold *irrespective of the fog server processor sharing policy.*

5.2 State of the Art

In the fog computing literature literature a few studies take into account the tradeoff between local processing and the cloud cost for the deployment of multi-modules applications. The objective here is to account for processor sharing effects on the placement of concurrent applications' chains in fog under the constraint of a limited budget for cloud usage.

Applications scaling and migration have been discussed extensively in the cloud literature [79, 67, 109, 113, 60]. Heuristic threshold policies have been extensively adopted before to solve feasibility problems [79, 67]. Reactive migration methods employ such thresholds to divert virtual machine instances from congested servers [109, 110]. Conversely, the threshold policies described in this work result from the minimisation of the processing delay. Load balancing in cloud, based on multi-server queue sampling has been studied, *e.g.,* in [113]. Our learning algorithm is based on a queueing sample scheme as well, but, it performs vertical load balancing between cloud and fog. Deterministic primal-dual algorithms appear, *e.g.,* in [60]. The stochastic solution proposed here converges to the optimal policy with imperfect state information leveraging noisy estimates of processing delays. In [66], authors proposed an online algorithm for service reconfiguration of edge-clouds; while considering limited storage capacity of edge servers, processor sharing effects are not accounted for. Wang et al. [105] studied dynamic edge service migration via Markov Decision Processes (MDPs) where migration depends on the desired service

location. In these two latter works, applications are represented as monolithic services without considering a more general structure consisting of interdependent modules.

Analytical models for resource sharing in cloud systems have been proposed in the literature [17, 101]. In particular, Processor Sharing (PS) is usually described as a *pre-emptive* policy where jobs can be stopped and resumed through their execution [64]. It has been deeply analysed in the queueing literature [12, 24, 13]. In that context, threshold policies have been identified as best responses for a game where customers can choose to be served on a private machine or on a remote mainframe [13]. A main difference with respect to those models is that in the proposed framework, applications incur a service rate slowdown since all the application queues run simultaneously on the same server without job interruption.

5.3 System Model

Let consider a set of N fog applications. Each application i is composed of n_i microservice modules. Batches of data are received by the first module of the i-th application at some rate λ_i and processed sequentially along the chain formed by the other modules downstream. For each application i, the first u_i modules can be placed in fog, whereas the rest of the chain, *i.e.,* the $n_i - u_i$ modules downstream, in cloud. Thus, $0 \leq u_i \leq n_i$ is a control variable representing the number of modules of application i placed on the fog server. Under policy $\mathbf{u} = (u_1, \ldots, u_n)$ the load of modules running on the fog server is $u = \sum_{i=1}^{N} u_i$.

When the fog server is serving u modules, a processor sharing scheme divides the computing resources among the modules running on the server. Let $G_i(u)$ be the sojourn time for a module of application i when the fog server hosts u modules, where the dependence on λ_i is omitted for the sake of notation. It indicates the time elapsing from the instant when a data batch is received from the tagged module till the end of processing, after which the resulting output is sent to the module downstream.

Processor sharing techniques reduce application's processing rate when multiple modules are hosted on the server. Every application module running on the fog server is subject to a stability condition, *i.e.,* there exists a critical load $u_{i,max}$ such that for each $u \geq u_{i,max}$, it holds $G_i(u) = +\infty$, and $G_i(u) < +\infty$ for $u \leq u_{i,max}$. Within their respective stability region $S_i = \{1 \leq u \leq u_{i,max}\}$, the $G_i(u)$s are assumed convex increasing in the fog server load.

Conversely, since a cloud system can provide a large number of servers, let d_i denote the constant average processing delay for modules of application i running in cloud. Finally, the processing delay of a data batch consumed by application i is given by the following governing equation

$$D_i(\mathbf{u}) = u_i G_i(u) + (n_i - u_i)\, d_i \tag{5.1}$$

In (5.1) the last $n_i - u_i$ modules of application i are executed in cloud: let c be the cost

paid to place a module on cloud. Also, the overall budget available to place modules in cloud is denoted $b_0 \geq 0$; $b := (\sum n_i - b_0/c) \geq 0$ is the minimum number of modules to be placed in fog. Finally, the cumulative delay $D(\mathbf{u}) := \sum_i D_i(\mathbf{u})$ is the target utility function used in the rest of the chapter.

Benchmark model: let consider an M/M/1 queue for the i-th application module. The service rate is μ_i data batches per second when the module runs alone on the server (let $\mu_i > \lambda_i$ to avoid trivial conditions). Conversely, when the server CPU is shared across application modules, applications continue processing in parallel but with a slowdown factor for the batch processing rate: under policy $\mathbf{u}$, service rate becomes $\mu_i(u) = \mu_i/u$ data batches per second. For the slowed M/M/1 case, the explicit expression for the batch processing delay writes $G_i(u) = (\frac{\mu_i}{u} - \lambda_i)^{-1}$. By direct calculations, the $G_i(x)$s, in this case, are convex increasing in the stability region. Note that, while this model is handy to explain the theoretical development, the results derived in the following apply in general to any convex increasing $G_i(u)$.

5.3.1 Validity of the model

Hereafter a few observations to precise the applicability of the proposed model.

- *Pipeline model*: the expression (5.1) is exact under the assumption that the modules of each application form a cascade of queues having all the same average batch processing delay as in Figure 5.1. E.g., in the benchmark model, when data batch arrival processes are independent, the first term of (5.1) represents the exact average processing delay of the i-th application due to the fog server [28]. The discussion in the following can be generalised to the heterogeneous case, *i.e.,* when the modules service times are not identical: for the sake of analysis, the discussion will be limited to the homogeneous case.

- *Processor Sharing (PS) model*: PS models have been extensively studied in the communication literature, capturing the situation where a single server processes jobs and reserves a fraction of the whole service time proportional to the number of customers [12]. The key difference, for instance, with the benchmark model is that each queue (*i.e.,* each application module) receives a fixed fraction of CPU clock: the u containers or virtual machines placed on the fog server incur into a u-fold slowdown of their data batch processing rate. However, it is worth observing for most PS models found in the networking literature, the expected system time, while very difficult to express analytically, appears invariably convex increasing in the server's load in all experiments, as per assumption on $G_i(\cdot)$.

- *Underlying MDP model*: in the rest of the chapter, $\mathbf{u}$ is a continuous control vector, where the placement of the last module in fog of application i occurs with probability $u_i - \lfloor u_i \rfloor$. Randomised threshold policies in at most one tap emerge naturally in

Table 5.1: Main notation used throughout the chapter

Symbol	*Meaning*
N	number of apps
b_0	budget to place apps in cloud
b	min. number of modules in fog $b := \sum n_i - b_0/c$
c	cost to place one app module in cloud
n_i	number of microservice modules for app i
u_i	number of modules in fog of app i
$\mathbf{u}$	placement policy $\mathbf{u} = (u_1, \ldots, u_N)$
u	fog server load $u = \sum_i u_i$
$u_{\max}$	maximum number of modules in fog
$G_i(u)$	proc. delay of app i modules fog batch at load u
μ_i	fog service rate for app i modules (M/M/1 PS)
λ_i	data batches per second towards app i (M/M/1 PS)
d_i	batch processing time in cloud for app i
D_i	batch processing delay for app i
D	cumulative processing delay $D := \sum_i D_i$

the theory of Markov Decision Processes [11]. Actually, governing equation (5.1) is still exact for integer values. But, since any randomised policy would perform a linear interpolation of deterministic policies, it would lead to a piecewise linear delay function. Conversely, relaxing the continuous control (5.1), the objective function introduced next, $\sum_{i=1}^{N} D_i(\mathbf{u})$, becomes a smooth convex interpolation, the tighter the larger the number of application modules.

5.4 Problem Formulation

The objective of the fog orchestrator is to optimally place the modules of each application in between a fog server and the cloud. The optimal placement policy can be determined according to the following formulation

Problem 5.1. Fog Placement

$$\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \sum_{i=1}^{N} D_i(\mathbf{u}) \\
\text{subject to} \quad & \sum_{i=1}^{N} c \cdot (n_i - u_i) \leq b_0 && (5.2) \\
& u_i > 0 \quad \textit{iff} \quad u \in S_i, \quad i = 1, \ldots, N && (5.3) \\
& 0 \leq u_i \leq n_i, \quad i = 1, \ldots, N && (5.4)
\end{aligned}$$

At this point, let precise the search space of the optimal solution. Since processor sharing techniques inevitably reduce the application's processing rate, constraint (5.3) forces the orchestration strategy to be such that all applications can complete batch processing within a finite time.

For the sake of concreteness, let refer to the benchmark model: there such constraint has the familiar form

$$\sum u_i < \frac{\mu_i}{\lambda_i}, \quad i = 1, \dots, N.$$

The processor sharing technique running on the fog server grants the stability condition for application i if and only if $u < \mu_i/\lambda_i$. Thus, when the search of an optimal solution $\mathbf{u}$ is performed in an interval where $u > \mu_i/\lambda_i$ for some application i, a candidate optimal solution needs to be such that $u_i = 0$, so that $D_i(\mathbf{u}) = d_i$ and the corresponding constraint can be removed. It follows immediately that the original problem maps into (at most) N subproblems, one for each partition induced by the stability condition (5.3), which can be solved in parallel to finally determine the optimal solution over the set of the (at most) N local minima. Each of such problems, as showed in the next section, is convex.

For the sake of simplicity, in the rest of the chapter such regions of instability are excluded, by assuming $u_{i,\max} \leq u_{\max}$, where $u_{\max} = \sum n_i$; in the case of benchmark model, this entails $\mu_i > \lambda_i u_{\max}$ for all $i = 1, \dots, N$. Hence, the analysis restricts to the case when $u \in [b, u_{\max}]$ so as to neglect bound (5.3).

The next section performs the general analysis of the problem and characterises the optimal solution. Before, the benchmark model for a single application serves as a concrete introduction to the general case.

5.4.1 Benchmark model for $N = 1$

From direct calculations on (5.1), in this case $D(u) = u/(\mu/u - \lambda) + d(n - u)$. Thus, the problem is apparently strictly convex in the stability region. Dropping all indexes for the notation's sake, the unique optimal fog placement control writes

$$u^* = \begin{cases} n & \text{if} \quad 0 < \lambda \leq \underline{\lambda} \\ \overline{u} & \text{if} \quad \underline{\lambda} < \lambda < \overline{\lambda} \\ b & \text{if} \quad \lambda \geq \overline{\lambda} \end{cases} \tag{5.5}$$

where $\overline{u} := \frac{\mu}{\lambda}(1 - (1 + \lambda d)^{-\frac{1}{2}})$ is the unique solution of the unconstrained minimization problem. Here, $\underline{\lambda}$ and $\overline{\lambda}$ are two thresholds: $\underline{\lambda}$ is the unique positive solution of $\frac{1}{\sqrt{1+xd}} = 1 - n\frac{x}{\mu}$ if $d < 2n/\mu_0$ or $\underline{\lambda} = +\infty$ otherwise; $\overline{\lambda}$ is the unique positive solution of $\frac{1}{\sqrt{1+xd}} = 1 - b\frac{x}{\mu}$ if $d < 2b/\mu$ or $\overline{\lambda} = +\infty$ otherwise; since $b \leq n$, then it holds $\underline{\lambda} \leq \overline{\lambda}$.

Even this simple case reveals a threshold structure varying with continuity with λ from very low batch arrival rates, where modules are all placed in fog, to high batch arrival rates, where it is optimal to place as many modules in cloud as possible, due to large delays introduced by the fog server.

5.5 Optimal Policy

Extending the result obtained for $N = 1$ to the general case, let identify a metric able to sort applications in order of importance, *i.e.,* represent the convenience of storing an application module either in cloud or in fog.

First, let characterise the convexity of the problem. Using auxiliary functions $u_i G_i(\sum u_i)$, the Hessian of $D(\mathbf{u})$ is positive definite according to the following result.

Theorem 5.1. *In the stability region* $\sum_{i=1}^{N} D_i(\mathbf{u})$ *is strictly convex so that Problem 5.1 has a unique solution.*

Proof. Indeed, $G_i(\sum u_i)$ is still convex since it is the composition of a convex function with an affine function. In order to verify the convexity of the objective function, one can verify the convexity of $D_i(\mathbf{u}) - d_i u_i = u_i G_i(\sum u_i)$. Without loss of generality, let restrict to $i = 1$, and define $f(\mathbf{u}) = G_1(\sum u_i)$ and $g(\mathbf{u}) = u_1$.

For the product fg of two real multivariate functions f and g, the Hessian writes

$$H(fg) = gH(f) + fH(f) + (\nabla g)^T \nabla f + (\nabla f)^T \nabla g$$

Here, it holds $\nabla f(\mathbf{u}) = \dot{G}(u)\mathbf{1}_n$, and $\nabla g(\mathbf{u}) = \mathbf{e}_1$, where $\mathbf{1} = (1, \ldots, 1)$ and $\mathbf{e}_1 = (1, 0, \ldots, 0)$, respectively. Finally, let write

$$H(u_1 G_1(u)) = u_1 H(G_1(u)) + \dot{G}_1(u)(\Lambda + \Lambda^T)$$

where $\Lambda = \mathbf{e}_1^T \mathbf{1}_n$. Since the spectrum $\sigma(\Lambda) = \{1, 0\}$, then $\Lambda \succeq 0$; indeed, $H(G_1(u)) \succeq 0$ because of the convexity of $G_i(u)$. Hence, it holds that for all is, $D_i(\mathbf{u})$ is convex, and it is strictly convex if $u_i > 0$. Thus, for all $\mathbf{u} \neq \mathbf{0}$, $D(\mathbf{u})$ is strictly convex; however, it needs to be strictly convex in all the domain $\mathbf{u} \geq \mathbf{0}$ as well, otherwise this would contradict the strict convexity in the one dimensional case. □

5.5.1 Marginal Delays

In order to find the structure of the optimal solution, the Lagrangian for the problem can be written as

$$\begin{aligned} L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma) \quad &= \sum_{j=1}^{N} u_j \Big(G_j(u) - d_j \Big) - \sum_{j=1}^{N} \alpha_j u_j \\ &\quad - \sum_{j=1}^{N} \beta_j (n_j - u_j) - \gamma(u - b) \end{aligned}$$

Constraint (5.3) does not appear since in the stability region the corresponding multipliers must vanish. For any policy $\mathbf{u}$ let define the key metric used in the rest of the chapter.

Definition 5.1. Marginal delay: *the marginal delay of application i under policy* $\mathbf{u}$ *is the parameter* $r_i(u) := G_i(u) - d_i$

The interpretation of this parameter is immediate: it is the difference of the batch processing delay in fog $G_i(u)$ and in cloud d_i. I.e., it measures the increase of processing delay when an application i is placed on the fog node instead of in cloud, under load u.

Marginal delays appear in the application of KKT conditions; here they are necessary and sufficient for Problem 5.1 since all constraints are affine [25]. First order KKT conditions write

$$L_{u_i} = r_i + \sum u_j \dot{G}_j(u) - \alpha_i + \beta_i - \gamma$$

The optimal solution will correspond to the set of nonnegative multipliers $\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}^*$ and γ^*. Previous relations and complementary slackness bring

$$\beta_j = -r_j + \gamma - \sum_{j=1}^{N} u_j \dot{G}_j(u),\ \alpha_j = 0, \quad \text{for } j = 1, \ldots, S$$

$$\alpha_j = r_j - \gamma + \sum_{j=1}^{N} u_j \dot{G}_j(u),\ \beta_j = 0, \quad \text{for } j = V+1, \ldots, N$$

where indexes are sorted such that $u_j = n_j$ for $j = 1, \ldots, S$ and $u_j = 0$ for $j = V + 1, \ldots, N$, respectively.

Now, for a given control vector $\mathbf{u}$, it is possible to sort conveniently the indexes based on complementary slackness conditions.

Proposition 5.1. *Let $r_1 \leq \ldots \leq r_S$ and $r_{V+1} \leq \ldots \leq r_N$, then $r_1 \leq \ldots \leq r_S \leq r_{S+1} = \ldots = r_V \leq r_{V+1} \leq \ldots \leq r_N$*

Proof. If $i < S < j < V$, let write $r_i + \beta_i = r_j$ and since $\beta_i > 0$, then $r_i < r_j$. In the same fashion, it is possible to write that, for $i < V < j$, it holds $r_i = r_j - \alpha_j$ and since $\alpha_j > 0$, then $r_i < r_j$. For $S + 1 \leq i \leq V$ follows from the fact that $\alpha_i = \beta_i = 0$. □

Since $S \leq V$, denote $S = V = 0$ to indicate that all application modules are placed in cloud, whereas $S = 0$ but $V > 0$ means no application has been fully placed in fog; $S = V > 0$ indicates that all applications are entirely placed either in fog or cloud.

Now it is possible to draw a few conclusions from Proposition 5.1:

1. For any optimal solution $\mathbf{u}^*$, the number of fog modules u^* induces the order according to which application modules should be placed in fog or cloud.

2. An optimal solution for which $\gamma = 0$, *i.e.,* $\sum u_i > b$, implies that the constraint is not active, *i.e.,* budget b_0 is not saturated. But, for such an optimal solution, it is always possible to replace the budget constraint such that $\sum u_i = b' > b$, *i.e.,* resorting to a case where the constraint is active, *e.g.,* $\gamma > 0$.

3. Once the structure of the optimal solution in the case $\gamma > 0$ has been determined, the optimal solution can be found in the one-dimensional space of parameters $b \leq u \leq u_{\max}$.

The above observations will be the basis for the algorithmic solutions solving Problem 5.1 which are proposed in the following sections.

5.5.2 Quasi-threshold structure

Hereafter, let further describe the optimal solution $\mathbf{u}^*$ for $\gamma > 0$: the result is a specific waterfilling type of policy, which becomes a threshold policy for all cases of interest.

Let assume the budget was saturated, then two indexes for the optimal solution S^* and V^* can be characterised by the following result:

Theorem 5.2. *Let the optimal solution* $\mathbf{u}^*$ *be attained for* $\gamma^* > 0$, *i.e.,* $u^* = \sum u_i = b$, *then*
i. $V^* = \min\{1 \leq j \leq N | \sum_{i=j+1}^{N} n_i \leq b_0\} := \overline{V}$
ii. Either $S^* = V^*$ *or* $S^* = V^* - 1$.

Proof. Case i. Let assume by contradiction that the optimal solution u^* is such that $V^* > \overline{V}$ (if $V^* < \overline{V}$ the constraint is violated). By employing an exchange argument, the uniqueness of the solution is contradicted. Let define $I := \{S^* + 1, \dots, V^*\}$: indeed $u_j < n_j$ for all $j \in I$, let construct a new policy which only differs for the indexes in set I as

$$\overline{u}_j = \begin{cases} u^*_{V^*} - \eta & j = V^* \\ u^*_j + \delta_j & j \in I \setminus \{V^*\} \\ u^*_j & j \notin I \end{cases} \tag{5.6}$$

where, since $V^* > \overline{V}$, indeed there exists positive η and δ_js such in a way that $\sum_{j \in I \setminus \{V^*\}} \delta_j = \eta$.

Let compare the delays under the two policies; preliminarily let observe that for any policy $\mathbf{u}$, it holds $D_i(\mathbf{u}) = r_i(u)u_i + n_i d_i$. Since $u^* = \overline{u} = b$, finally

$$\begin{aligned} \Delta D = D(\mathbf{u}^*) - D(\overline{\mathbf{u}}) &= \sum_{i \in I} D_i(\mathbf{u}^*) - D_i(\overline{\mathbf{u}}) \\ &= r_{V^*}\eta - \sum_{i \in I \setminus \{V^*\}} r_i \delta_i \geq r_{V^*}\Big(\eta - \sum_{j \in I} \delta_j\Big) = 0 \end{aligned}$$

where the last step, due to the fact that the marginal delays are sorted non decreasing, concludes the proof of part i.

Case ii. First, from the definition of I, it holds $0 < u_i < n_i$ for all $i \in I$. If $S^* = V^*$, then the budget constraint is saturated

$$\sum_{i \in I} (n_i - u_i) = b_0 - \sum_{j=V^*+1}^{N} n_j = 0$$

so that $I = \emptyset$. Conversely, let $S^* < V^*$. If $S^* = V^* - 1$ there is nothing to prove. Consequently, if $|I| > 1$ let I be sorted according to increasing values of r_i. The idea is

to show that we can build a new optimal solution with at most one application deployed between the fog server and the cloud, *i.e.,* a solution for which $|I| = 1$ and hence $S^* = V^* - 1$. Since $|I| > 1$ we can assume that there exist two applications deployed between fog and cloud, i and j. Their contribution to the total delay is

$$f(u_i, u_j) := u_i\,(G_i(u) - d_i) + u_j\,(G_j(u) - d_j) + n_i\,d_i + n_j\,d_j, \tag{5.7}$$

with $0 < u_i < n_i$ and $0 < u_j < n_j$. We indicates with $\bar{u}$ the sum of u_i and u_j, i.e., $\bar{u} = u_i + u_j$. Now we show that we can build a better solution with at most one application between cloud and fog. We want to minimize function (5.7) keeping the sum of u_i and u_j constant. That is

$$\begin{array}{ll} \text{minimise} & f(u_i, u_j) \\ \text{subject to} & u_i + u_j = \bar{u} \\ & u_i \in \{0, \ldots, n_i\}, \quad u_j \in \{0, \ldots, n_j\}. \end{array}$$

Assuming $G_i(u) - d_i < G_j(u) - d_j$, we should increase the number of fog modules of the application with the lowest delay. Indeed

$$\begin{aligned} u_i\,[G_i(u) - d_i] + u_j\,[G_j(u) - d_j] = \\ u_i\,[G_i(u) - d_i] + (\bar{u} - u_i)\,[G_j(u) - d_j] = \\ u_i\,[G_i(u) - d_i - G_j(u) + d_j] + \bar{u}\,[G_j(u) - d_j]. \end{aligned} \tag{5.8}$$

From equation (5.8) follows that the value of u_i must be increased in order to minimize (5.7). Hence, to keep $\bar{u}$ constant, the value of u_j should be decreased. Now we have two cases:
a) $u_j \geq n_i - u_i$. Hence, $u'_i = n_i$, i.e., application i is entirely deployed in fog, and $u'_j = n_j - (n_i - u_i)$. In this case we have

$$D(u_i, u_j) \geq f(n_i, u'_j) > D(n_i, u'_j),$$

where $D(u_i, u_j)$ is the delay function with values u_i and u_j for application i and j, respectively.
b) $u_j < n_i - u_i$. Hence, $u'_j = 0$ and $u'_i = u_i + u_j$. In this case, the application j is entirely placed on cloud and the application i deployed between the cloud and the fog. The new configuration respects the following inequality

$$D(u_i, u_j) \geq f(u_i + u_j, 0) > D(u_i + u_j, 0).$$

In both cases, we have built another configuration with at most one application deployed between cloud and fog with a non-increasing delay with respect to the previous configuration, contradicting our initial assumption. □

The actual structure of the optimal solution is now derived. In order to simplify the discussion, with no loss of generality, wherever $r_i = r_j$, the indexes in I will be sorted for nonincreasing values of $n_i d_i$. The key role of the order of the r_is induced by u is reflected in the following:

Definition 5.2. Marginal delays order. *For any value of* $b \leq x \leq u_{\max}$, $u_{\max} := \sum n_i$, *let* $\sigma_x : \{1, \ldots, N\} \to \{1, \ldots, N\}$ *be the permutation ordering of the marginal delays such that* $r_{\sigma_x(1)} \leq \ldots \leq r_{\sigma_x(N)}$ *and* $n_{\sigma_x(i)} d_{\sigma_x(i)} \geq n_{\sigma_x(i+1)} d_{\sigma_x(i+1)}$ *in case equality holds. Denote* $\Sigma = \{\sigma_u | b \leq u \leq b_{\max}\}$ *the set of such permutations in the stability region.*

The role of this definition is needed in the proof of the following:

Theorem 5.3. *Let* $\gamma^* > 0$, *and assume* $u^* = b$. *Then the optimal solution* $\mathbf{u}^*$ *has the following structure*

$$u_i^* = \begin{cases} n_i & \text{if} \quad 1 \leq i \leq S^* \\ \frac{\gamma^* - r_{V^*} - \sum_{j=1}^{S^*} n_j \dot{G}_j(b)}{(V^* - S^*)\dot{G}_i(b)} & \text{if} \quad S^* + 1 \leq i \leq V^* \\ 0 & \text{if} \quad V^* < i \leq N \end{cases} \tag{5.9}$$

where $V^* = V^*(b)$ and $S^* = S^*(b)$ are as in Thm. 5.2 and the order of the indexes is determined by σ_b.

Proof. The statement needs to be proved for $S^* + 1 < i \leq V^*$ only. First, let observe that $\dot{G}_i(u) > 0$. Second, the solution is obtained by solving the following system

$$\sum_{j=S^*+1}^{V^*} u_j \dot{G}_j(b) = \gamma - r_{V^*} - \sum_{j=1}^{S^*} n_j \dot{G}_j(b)$$
$$\sum_{j=S^*+1}^{V^*} u_j = b - \sum_{j=1}^{S^*} n_j$$

Hence, a solution for the first equation can be obtained in the form

$$u_i^* = \frac{\gamma - r_{V^*} - \sum_{j=1}^{S^*} n_j \dot{G}_i(b)}{(V^* - S^*)\dot{G}_i(b)}$$

where, in order to satisfy the saturation condition, it is always possible to define

$$\gamma^* = r_{V^*} + \sum_{j=1}^{S^*} n_j \dot{G}_j(b) + \frac{(V^* - S^*)(b - \sum_{j=1}^{S^*} n_j)}{\sum_{j=S^*}^{V^*} 1/\dot{G}_j(b)}$$

concluding the proof, since the solution is unique. □

Finally, when $\text{card}(I) = 1$, the solution is a threshold policy with at most one non-extremal control $u_{V^*} = b - \sum_{j=1}^{V^*-1} n_j$. In this case, indeed, the calculation of γ^* is not needed in order to determine the optimal solution. Conversely, when $\text{card}(I) > 1$, the policy is "almost threshold", *i.e.,* it is extremal for all $i \notin I$. For the sake of notation, in the rest of the discussion, we assume $n_i d_i \neq n_j d_j$ for all $i, j = 1, \ldots, N$; next results can be extended to the case when equality holds for some index.

5.6 Algorithmic Solution

In order to fully determine the optimal solution, it is necessary to determine the actual value of the budget under which the constraint is saturated and optimal. The idea is to operate a search in the space of the threshold policies in the form determined in Theorem 5.3, parametrised in the one-dimensional domain of the actual budget spent. First, it can be proved the cardinality of the set of permutations Σ induced by the order of the r_is is polynomially bounded by the number of applications due to the monotonicity and convexity of marginal delays, $|\Sigma| = O(N^2)$. Hence, with this remark, it is possible to partition the interval $[b, u_{max}]$ in subintervals defined by each permutation of the r_is within the stability region, $[b, u_{max}] = \cup_{k=1}^{K-1} A_k$, where K is the maximum number of intersections between two different marginal delay functions. In this manner, proving that the objective function is piecewise convex in $u \in A_k$, for each k, it is possible to perform a bisection search on each interval computing the placement policy with the minimum delay on that interval. Finally, the minimum among all the intervals is taken. This permits to solve the original problem with complexity $O(N^2)$ in the worst case.

Let remark first on the cardinality of the set of permutations Σ introduced in Definition 5.2. Due to monotonicity and convexity, the number of permutations in Σ induced by the order of the r_is is expected to be finite for most practical cases. For the benchmark system, the marginal delay of two different applications can at most attain equality for $k = 2$ values of the load u, simply inspecting equation

$$\frac{u}{\mu_i - u\lambda_i} - d_i = \frac{u}{\mu_j - u\lambda_j} - d_j.$$

More generally, the following bound on the number of search interval holds:

Lemma 5.1. *Let equation $r_i(u) = r_j(u)$ have at most k solutions, then $|\Sigma| \leq k\binom{N}{2}$.*

Proof. For every $b \leq u \leq b_{\max}$, the order of the r_is is induced by verifying $r_i(u) \leq r_j(u)$ is true or false for every pair of applications $i \neq j$. Starting at $u = a$, the order of the $r_i(u)$s can change at most $k\binom{N}{2}$ times, from which the statement follows. □

Since Σ has cardinality at most quadratic in the number of applications N, it can thus be computed in polynomial time. Furthermore, from the previous result, let partition $[b, u_{\max}] = \cup_{k=1}^{K-1} A_k$ where $[a_k, a_{k+1}]$, $a_1 = b$ and $a_K = u_{\max}$. Clearly, σ_u is invariant

within each interval of the partition, *i.e.,* $\sigma_u = \sigma_{a_k}$ for $u \in [a_k, a_{k+1}]$. Thus, σ_k indicates permutation σ_u over interval A_k.

5.6.1 Threshold policy decomposition

For each permutation $\sigma_k \in \Sigma$, and for every value $b \leq u \leq u_{\max}$, let consider policy $\mathbf{u}(\sigma_k, u)$ constructed as follows:

$$u_i(\sigma_k,u):=\begin{cases} n_i & \text{if} \quad u\geq\sum_{j=1}^{i} n_{\sigma_k(i)} \\ u-\sum_{j=1}^{i-1} n_{\sigma_k(j)} & \text{if} \quad \sum_{j=1}^{i-1} n_{\sigma_k(j)}\leq u<\sum_{j=1}^{i} n_{\sigma_k(j)} \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

The policies designed in (5.10) coincide with the ones in Theorem 5.2 for all values of $u \in A_k$. Of course, each such policy, in general, is sub-optimal outside the interval A_k. The next immediate result shows that the search for the optimal solution can be restricted to the set of policies in (5.10):

Lemma 5.2. *Let* $\mathbf{u}^*$ *be the optimal solution. Let define*

$$(\widehat{\sigma}, \widehat{u}) = \arg \min_{b \leq u \leq u_{\max}} D(\mathbf{u}(\sigma, u))$$

and the corresponding policy $\widehat{\mathbf{u}} = \mathbf{u}(\widehat{\sigma}, \widehat{u})$*. Then* $\mathbf{u}^* = \widehat{\mathbf{u}}$*.*

Proof. First, let observe that since $\mathbf{u}^*$ is attained at some value $u^* = \sum u_i^*$, indeed $\widehat{\mathbf{u}} = \mathbf{u}(\sigma_{u^*}, u^*)$, so that $D(\mathbf{u}(\widehat{\sigma}, \widehat{u})) \leq D(\mathbf{u}(\sigma_{u^*}, u^*))$ from (5.11). But, from the optimality of u^* it must hold, $D(\mathbf{u}(\widehat{\sigma}, \widehat{u})) \geq D(\mathbf{u}(\sigma_{u^*}, u^*))$. Since by construction both solutions respect the constraints, from the uniqueness of the optimal solution $\mathbf{u}^* = \widehat{\mathbf{u}}$. □

Finally, in order to discuss the complexity of our proposed algorithm the following statement is needed

Theorem 5.4. *Fixed* $\sigma_k \in \Sigma$*, the function* $D(\mathbf{u}(\sigma_k, u))$ *is piecewise convex in* $u \in A_k$*.*

Proof. Let define the set of switching points $\{v_s\}$ where, if $u = v_s$, the s-th application is fully placed in fog, *i.e.,*, $v_s = \sum_{j=1}^{s} n_{\sigma_k(j)}$. Let define $B_{k,s} := [v_s - 1, v_s] \cap A_k$, where $v_0 = a_k$ and $v_H = a_{k+1}$. Now, let partition the interval $A_k = \cup_{s=1}^{H} B_{k,s}$, and for $u \in V_s$ it follows

$$D(\mathbf{u}(\widehat{\sigma}, \widehat{u})) = C + \sum_{j=1}^{s-1} n_{\sigma_k(j)} G_{\sigma_k(j)}(u) + (u - v_{s-1}) G_{\sigma_k(s)}(u)$$

where C is a constant from which the statement follows □

```
MDTA({G_i}, d)
Precalculate: Σ = {σ_k}, {A_k} and {B_s}
Initialize: D* ← ∞, D_mid ← ∞
            u* ← 0, u_mid ← 0
1: FOR k = 1, ... K
2:     σ ← σ_k
3:     FOR s = 1, ... H
4:         u_L = b_{s-1}, u_R = b_s,
5:       WHILE |u − u_tmp| > ε
6:           u_mid ← (u_L + u_R)/2
7:           u_mid ← u(σ, u_mid)
8:           IF  ∂_u D(u_mid) < 0
9:               u_L ← u_mid
10:           ELSE
11:              u_R ← u_mid
12:          END
13:       END
14:       IF  D(u_mid) < D*
15          u* ← u_mid
16          D* ← D(u_mid)
17:       END
18:    END
19: END
20: RETURN u* and D*
```

Figure 5.2: Pseudocode of the MDTA algorithm.

It is now possible to describe the Marginal Delays Threshold Algorithm (MDTA). The pseudocode of MDTA is reported in Figure 5.2: it works by searching over the space of the budget $[b, u_{\max})$ and returns the optimal delay and the corresponding optimal fog placement. It receives as input the specifications of each application i, the processing delay $G_i(\cdot)$ and its cloud service time d_i; for the benchmark model, this amounts to receive the arrival rate λ_i and its fog service rate μ_i. MDTA performs the precomputation of the partition $\{A_k\}$, the corresponding permutation σ_k, and the subintervals $\{B_{k,s}\}$ defined in Theorem 5.4. Then, for every interval A_k (FOR cycle starting at line 1), it explores the threshold policy within every sub-partition $\{B_{k,s}\}$ of A_k (FOR cycle starting at line 3). From Thm. 5.4, within each $B_{k,s}$ the $D(\mathbf{u}(\sigma, u))$ is convex: a bisection search applies (line 5 to 13), where the sign of the subgradient $\partial_B D$ guides the algorithm to move to the right half of the domain or to the left. Finally the minimum is found exploring all possible intervals (line 14 to 18).

5.6.2 Complexity Analysis

While the problem analysis was meant to describe the structure of the optimal solution, state-of-the-art tools for convex optimization can be used to solve Problem 5.1; however, general interior point methods and ϵ-accuracy methods rarely provide polynomial complexity bounds in the input size so that their scalability is debated [85]. The proposed solution has worst case complexity which is quadratic in the input size of the problem with a parallel implementation. However, as showed in the numerical section, the $O(N^2)$ estimation for the number of intervals σ is rather conservative and it appears almost linear in the input size.

Furthermore, the pseudocode in Figure 5.2 is a sequential implementation with complexity $O(N^2 \log(\frac{u_{\max}}{\epsilon}))$. With a simple parallelisation, and observing that, for each A_k, the number of subintervals $\{B_{k,s}\}$ is $O(1)$ since it is upper bounded by $u_{\max} - b$, it holds

Theorem 5.5. *MDTA has complexity $O(N^2)$.*

Proof. From Lemma 5.1, K, *i.e.,* the maximum number of intersections between two different marginal delay functions, is $O(N^2)$. The number of subintervals $\{B_{k,s}\}$ corresponds to the number of switches possible within A_k, *i.e.,* the maximum number of modules that can be placed in fog. However, such number is upper bounded by $u_{\max} - b$, which is an input of the problem and not dependent on N. The bisection search can be performed in parallel for each interval, and all the $K(u_{\max} - b)$ local minima should be compared. □

5.6.3 Example

Figure 5.3 reports on the dynamics of the r_is for an example for $N = 3$ with $n_1 = 8$, $n_2 = 9$ and $n_3 = 7$ modules, respectively. The order on the r_is is determined by the intersections $r_i(u) = r_j(u)$. The exploration space is partitioned according to the sets $\{A_k\}$, *i.e.,* three valid intervals $A_k = [a_k, a_{k+1}]$, where $a_1 = b$, a_2 solves for $r_2(a_2) = r_3(a_2)$, a_3 solves for $r_1(a_3) = r_3(a_3)$ and $a_4 = u_{\max}$. In the example $b_0 = 24$, hence the minimum number of modules to be deployed on fog is $b = 0$: the optimal solution u^*, highlighted by vertical red line, lies in the first interval. Observe that $u^* > b$: the solution identifies the optimal fraction of the budget to be spent.

5.7 Online Learning

In much part of the current literature, cloud and fog orchestration is performed with an initial placement, typically based on nominal load values, and using later online migration techniques to reduce hotspots at runtime [109, 110]. In fact, several system parameters may change over time: batch arrival rates $\{\lambda_i\}$, applications' fog processing delays $\{G_i\}$ and cloud processing delays $\{d_i\}$ may fluctuate around their nominal values or drift, *e.g.,* due to variations in fog applications' workloads. They might even switch to different

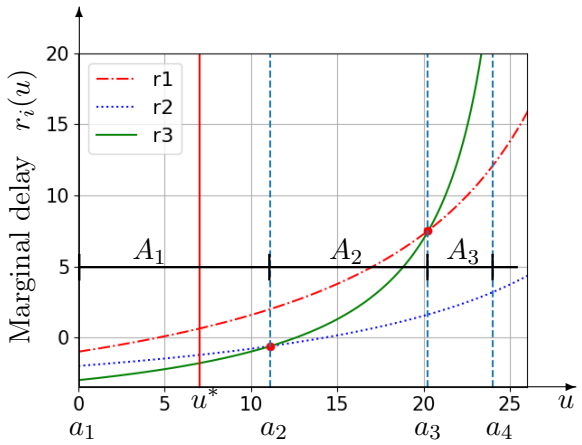

Figure 5.3: Example for $N = 3$ with $\boldsymbol{\lambda} = (0.14, 0.26, 0.3)$ s^{-1}, $\boldsymbol{\mu} = (5.33, 10.9, 7.96)$ s^{-1}, $\boldsymbol{d} = (1, 2, 3)$ s, $n_1 = 8$, $n_2 = 9$, and $n_3 = 7$.

values as a consequence of sudden changes in operating conditions (*e.g.,* objects data generation rates may change). As a result, the $\{r_i\}$s, considered so far as deterministic quantities, form in fact a random process. Let assume that the system is configured in some interior point of the domain $\mathbf{x}_0 > 0$: this is sufficient to obtain, for each tagged application i, the estimates $\{\widetilde{G}_{i,n}\}_{n\in\mathcal{N}}$ and $\{\widetilde{d}_{i,n}\}_{n\in\mathcal{N}}$ for the batch processing time in fog and in cloud, respectively. Such samples are generated at rate λ_i. If this is not the case, for the next scheme to work it is necessary to introduce probing schemes able to migrate some modules to the fog or the cloud in order to obtain the needed samples. However, such schemes are out of the scope of the present work.

Hereafter, based on the structure of the optimal solution, a continuously adaptive procedure is designed. It works under the assumption that an optimal policy saturates the available budget, *i.e.,* $\sum n_i - u_i^* = b_0$. The algorithm performs the optimization of marginal delays

Problem 5.2. Marginal Delay Optimization (MDO)

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & -\sum_{i=1}^{N} r_i \cdot n_i \cdot x_i \\ \text{subject to} \quad & \sum_{i=1}^{N} n_i \cdot x_i - b_0 \leq 0 \\ & 0 \leq x_i \leq 1, \quad i = 1, \ldots, N \end{aligned}$$

where the identification is for the sake of notation $x_i = 1 - u_i/n_i$, obtaining a fractional knapsack problem [72]. It is easy to verify that the optimal solution solves Problem 5.1 under budget saturation. However, let assume to have just noisy measurements of both the objective function and the constraints. A tool to handle this situation is represented

by stochastic approximation [74]. Hence, hereafter a stochastic primal-dual optimization algorithm of the family discussed in [73] is introduced. This entails a learning procedure of the Robinson-Monroe type, which is known to have efficient noise rejection properties [74]. However, it is convenient to replace the objective function in Problem 5.2 with a convex one as follows

Problem 5.3. Convexified MDO

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) := -\sum_{i=1}^{N} r_i \cdot n_i \cdot \frac{1-e^{-x_i}}{1-e^{-1}} \\ \text{subject to} \quad & q(\mathbf{x}) := \Big(\sum_{i=1}^{N} n_i \cdot x_i - b_0\Big) \leq 0 \\ & 0 \leq x_i \leq 1, \quad i = 1, \ldots, N \end{aligned}$$

It is immediate to observe that the optimal control attained by Problem 5.3 coincides with the solution of Problem 5.2 (even if the value of the attained minimum is different). The online learning procedure is based on the Lagrangian associated to Problem 5.3, thus minimising

$$L(\mathbf{x}, \theta) = f(\mathbf{x}) + \theta\, q(\mathbf{x}) - \boldsymbol{\xi} \cdot \mathbf{x} + \boldsymbol{\nu} \cdot (\mathbf{x} - \mathbf{1}), \tag{5.11}$$

where $\boldsymbol{\xi} \geq \mathbf{0}$ and $\boldsymbol{\nu} \geq \mathbf{0}$ are the multiplier vectors accounting for the box constraints $0 \leq x_i \leq 1$, for $i = 1, \ldots, N$, and $\theta \geq 0$ accounts for the coupled constraint. The iteration for the update of the primal and of the dual variables writes as follows

$$\begin{aligned} x_{i,n+1} &= \Pi_{[0,1]}\left[x_{i,n} - \varepsilon_n \widetilde{L}_{x_i}(\mathbf{x}_n, \theta_n)\right] \\ \theta_{n+1} &= \Pi_{\geq 0}\left[\theta_n + \varepsilon_n q(\mathbf{x}_{n+1})\right] \\ \xi_{i,n+1} &= \Pi_{\geq 0}\left[\xi_{i,n} - \varepsilon_n x_{i,n+1}\right] \\ \nu_{i,n+1} &= \Pi_{\geq 0}\left[\nu_{i,n} + \varepsilon_n (x_{i,n+1} - 1)\right], \end{aligned} \tag{5.12}$$

where $\{\varepsilon_n\}_{n \in \mathcal{N}}$ are the stepsizes of the algorithm, and obey to the following assumption

$$\varepsilon_n \geq 0, \quad \sum_{n=0}^{+\infty} \varepsilon_n = +\infty, \quad \sum_{n=0}^{+\infty} {\varepsilon_n}^2 < +\infty. \tag{5.13}$$

In (5.12), $\Pi_{[0,1]}(y) = \max(0, \min(y, 1))$ and $\Pi_{\geq 0}(y) = \max(0, \min(y))$ denote projection functions.

Let $\widetilde{L}_{x_i}$ indicate noisy estimates of the k-th component of the gradient of the Lagrangian: each time an estimate is produced, the i-th control component is updated, and the dual variables θ, $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$ as well. The update instants triggering the updates in (5.12) are those when a new measurement of the marginal delay of application k is available. Actually, the explicit expression appearing in (5.12) is

$$\widetilde{L}_{x_i}(\mathbf{x}_n, \theta_n) = n_i \left(\frac{e}{e-1}\big(\widetilde{G}_i - \widetilde{d}_i\big)e^{-x_{i,n}} - \theta_n\right) - \xi_{i,n} + \nu_{i,n}$$

where $\widetilde{G}_i$ and $\widetilde{d}_i$ are estimates of the system time of modules in fog and in cloud.

The following result ensures the convergence to the unique solution of Problem 5.2.

Theorem 5.6. *Let sequence* $\{\varepsilon_n\}$ *satisfy* (5.13) *and let the* $\widetilde{G}_i$ *and* $\widetilde{d}_i$ *be unbiased estimates with finite second order moments. Then the sequence of policies* $\mathbf{x}_n$ *converges to the optimal policy* $\mathbf{x}^*$ *with probability one.*

Proof. The proof is based on the ODE method and requires to verify first the assumptions of Theorem 2.1 in [74]. Let consider the update equation for the component $x_{i,n}$: same relations are verified in the same way for the other components, and are omitted for the sake of space. Let rewrite the update equation (5.12) for $x_{i,n}$ in the form $x_{i,n+1} = x_{i,n} + \varepsilon_n(Y_n + Z_n)$, where

$$Y_{i,n} = L_{x_i}(\mathbf{x}, \theta, \boldsymbol{\xi}, \boldsymbol{\nu}) + \delta M_n + \beta_n \tag{5.14}$$

In (5.14), the term β_n allows for the presence of an asymptotically unbiased error, Z_n is the error due to projection, while δM_n is the martingale noise estimation, *e.g.,*

$$\delta M_n = Y_n - \mathbb{E}_n[Y_n|(\mathbf{x}_0, \theta_0, \boldsymbol{\xi}_0, \boldsymbol{\nu}_0), Y_i, i < n]$$

where index 0 denotes the initial conditions.

The assumptions of Theorem 2.1 in [74] are verified hereafter with respect to the problem:
A1) $\sup_n \mathbb{E}|Y_n|^2 < +\infty$: this condition is immediately verified since $\widetilde{G}_i$ and $\widetilde{d}_i$ have finite second order moments;
A2) There is a measurable function $\overline{g}(\cdot)$ of $(\mathbf{x}, \theta)$ and random variables β_n such that $\mathbb{E}_n[Y_n|\mathbf{x}_0, Y_i, i = 1 < n] = \overline{g}(\mathbf{x}, \theta) + \beta_n$: by construction it holds

$$\overline{g}(\mathbf{x}, \theta) := (\nabla_x L(\mathbf{x}, \theta), q(\mathbf{x}), \mathbf{x}, \mathbf{1} - \mathbf{x}),$$

which is continuous and thus measurable; also, by assumption estimates are unbiased and the gradient of the Lagrangian is linear in the estimates so that $\mathbb{E}_n \delta M_n = 0$ and $\beta_n = 0$ for all $n \in \mathcal{N}$;
A3) $\overline{g}(\mathbf{x}, \theta)$ needs to be continuous: because the Lagrangian is continuously differentiable this assumption indeed holds;
A4) (5.13) is satisfied: by assumption;
A5) $\sum \varepsilon_n |\beta_n| < \infty$ with probability 1: true since $\beta_n = 0$ for all $n \in \mathcal{N}$.
Once the above assumptions hold true, the trajectories of (5.12) are guaranteed to converge to an invariant set of the ODE identified by the equations

$$\begin{aligned} \dot{\mathbf{x}} &= \nabla_x L(\mathbf{x}, \theta, \boldsymbol{\xi}, \boldsymbol{\nu}), \quad \dot{\theta} = q(\mathbf{x}) \\ \dot{\boldsymbol{\xi}} &= -\mathbf{x}, \quad \dot{\boldsymbol{\nu}} = \mathbf{x} - \mathbf{1}. \end{aligned}$$

As stated by Theorem 2.1 in [74], if the constraint set is dropped but the trajectories of (5.12) are bounded with probability one, we still have the guarantee of convergence

to an invariant set of (5.15). Furthermore, since the ODE is Lipschitz, the solution is unique. But, a restpoint of the above equation solves for the first order KKT conditions of Problem 5.3 and verifies also all complementary slackness conditions. Hence, it must coincide with the optimal solution. Thus the invariant set is, by construction, the optimal policy and the multipliers associated with it. Theorem 2.1 in [74] grants that the sample paths of the algorithm converge with a certain probability to such restpoint. The asymptotic stability of the restpoint with respect to (5.15) grants that the sample paths of the algorithm converge a.s. to the restpoint of (5.12). Actually, as proved in [38][Lemma 4.1], the strict convexity of the Lagrangian grants that the Lyapunov condition for asymptotic stability of a restpoint is verified for the primal-dual trajectory in (5.15). This concludes the proof. □

From the proof of Theorem 5.6, the learning procedure works with any possible estimator providing unbiased samples of an application's processing delays. E.g., the sample mean of the elapsed time between the time when a batch is fed to a module and the time when the output result is given as input to the module downstream.

Finally, in the numerical section it is employed a penalty function $p(\mathbf{x}) =: \frac{1}{2}\, p_0\, q^2(\mathbf{x})$ [19], where p_0 is a tunable penalty factor. While this operation does not change the optimal solution, a notable increase of the numerical stability of the algorithm is observed; the proof of convergence can be adapted in the obvious way.

5.7.1 Adaptive Version

In order to adapt faster to changing parameters, it is possible to use a variant of the stochastic approximation technique where the step of approximations has small but constant size. Using the same algorithm, the constant stepsize $\varepsilon_n = \varepsilon > 0$ is used for all n: because the approximation step does not vanish over time, the system continues to adapt.

However, while for decreasing stepsizes convergence in probability to the solution of the KKT conditions is proved, for constant stepsizes convergence results are weaker. In particular, let consider neighborhoods of radius δ around the optimal solution $\boldsymbol{\varphi}^* = (\mathbf{x}^*, \theta^*, \boldsymbol{\xi}^*, \boldsymbol{\nu}^*)$ of the type $N_\delta(\boldsymbol{\varphi}^*) = \{\boldsymbol{\varphi} \in \mathbb{R}^{3N+1} : ||\boldsymbol{\varphi} - \boldsymbol{\varphi}^*||_2 < \delta\}$. The following result grants that, as long as a sufficiently small stepsize is chosen, the algorithm produces an output which remains confined around a suitable neighborhood of the optimal solution

Theorem 5.7. *For any $\delta > 0$, define by $N_\delta(\boldsymbol{\varphi}^*)$: for $\epsilon \to 0$, the sample paths of the algorithm defined in the iteration* (5.12) *for constant stepsize converge in distribution to elements in $N_\delta(\boldsymbol{\varphi}^*)$. The fraction of time spent by the process in $N_\delta(\boldsymbol{\varphi}^*)$ during $[0, T]$ goes to 1 as time horizon T diverges.*

The above result can be proved by verifying that Theorem 2.1 at pp. 248 in [74] holds true.

5.8 Numerical Results

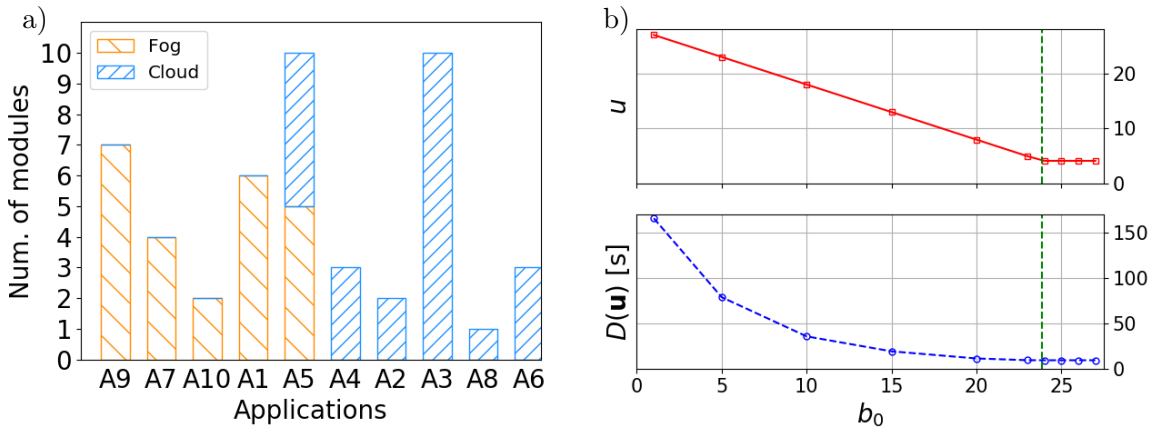

Figure 5.4: a) Threshold structure of the optimal solution; $N = 10$, $n_i \in [1, 10]$, $\lambda_i \in [0.1, 0.3]$ s^{-1}, $\mu_i \in [5, 16]$ s^{-1}, $d_i \in [0.5, 3.3]$ s; b) Optimal control and cumulative delay for increasing budget b_0; $N = 10$, $n_i \in [1, 5]$, $\lambda_i \in [0.1, 0.3]$ s^{-1}, $\mu_i \in [5, 15]$ s^{-1}, $d_i \in [0.5, 3.3]$ s.

In this section the characterisation of the optimal fog orchestration policy is completed by performing numerical exploration. Figure 5.4a describes the optimal policy for a set of $N = 10$ applications adhering to the benchmark model, where the parameters of applications are chosen uniformly at random in their respective intervals, *i.e.,* $n_i \in [1, 10], \lambda_i \in [0.1, 0.3]$ s^{-1}, $\mu_i \in [5, 16]$ s^{-1}, $d_i \in [0.5, 3.3]$ s. Applications' indexes have been sorted according to the r_is corresponding to the optimal solution produced by Matlab® nonlinear optimization toolbox. The optimal policy is clearly of threshold type. Furthermore, Figure 5.4b depicts the behaviour of the optimal control and the cumulative delay at the increase of the budget available for the offloading to the cloud. It is seen there that there exists a certain threshold ($b_0 = 23.8$ in the figure): above such value, in fact, the aggregated delay does not improve by further offloading to the cloud, *i.e.,* with larger costs. Rather, the local server provides a computational advantage for a subset of the applications under moderate load. Figure 5.4b confirms that, as a byproduct of the optimization, one can obtain a precise answer to the practical dilemma whether or not saturating the available budget is optimal, whose answer is not obvious apriori from the system parameters. The figure also confirms that an identically null solution, $u = 0$, cannot be an optimal solution. Hence, even though there is enough budget to put all the applications on the cloud and low delays in cloud, the optimal solution never takes into account this option when, as in the experiment, $r_i(1) < 0$ for some application i.

Figure 5.5a describes the comparison, for increasing budget size, of the optimal solution obtained via the optimization toolbox (Opt), the `MDTA` algorithm and a benchmark greedy, load-based fog orchestration algorithm (greedy). The last algorithm sorts the applications

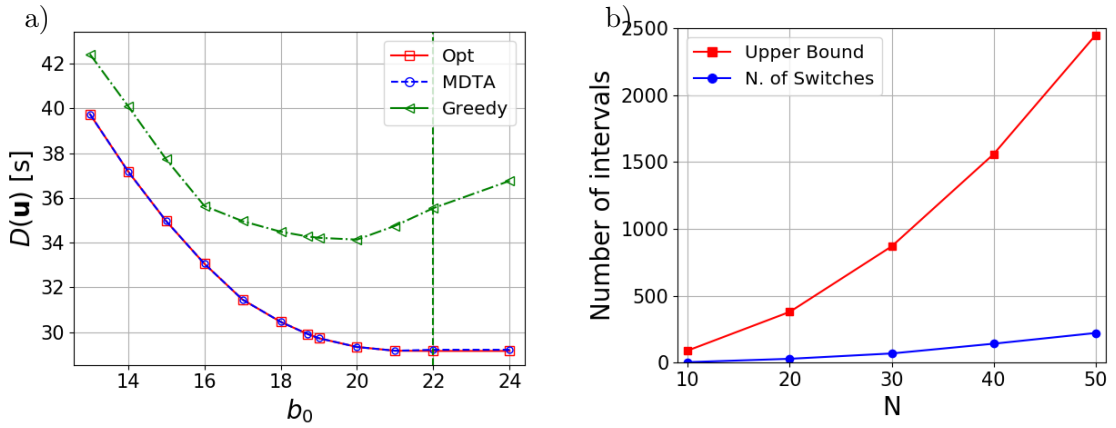

Figure 5.5: a) Cumulative delay attained by Opt, `MDTA`, and greedy algorithm, respectively, for increasing budget b_0; b) Number of intervals $\{A_k\}$ and upper bound.

based on the nominal load of the i-th application, *i.e.*, the ratio λ_i/μ_i, but neglects the effect of the processor sharing on the fog server. As proved in the theoretical development, `MDTA` attains indeed the optimal solution. The greedy algorithm presents a significant performance loss with respect to the optimal policy for the same budget.

In the worst-case complexity analysis, the computational complexity of `MDTA` has been dominated by the number of the intervals A_k. Fig. 5.5b provides a numerical evaluation of the actual number of the intervals A_k. From that experiment the quadratic upperbound on the number of intervals appears conservative, and it is possible to conjecture that the complexity of `MDTA` might result moderately superlinear in the input size on average.

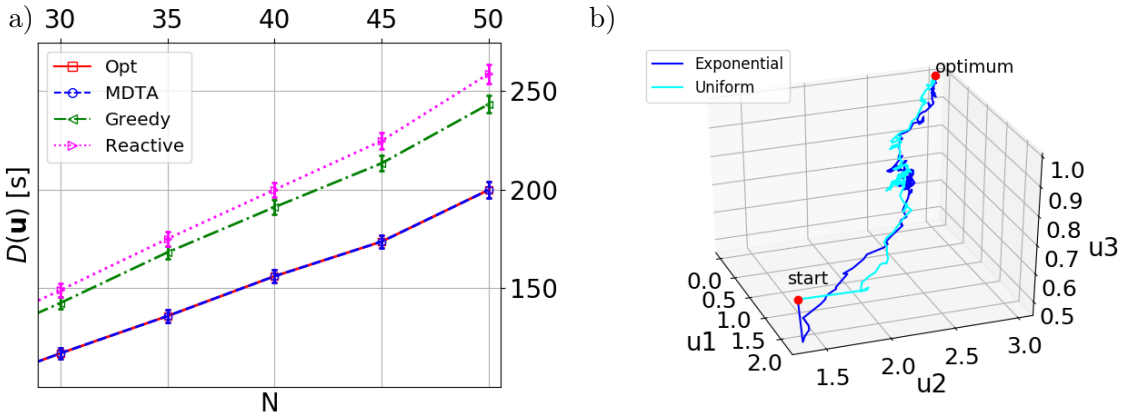

Figure 5.6: a) Cumulative delay for Opt, `MDTA`, greedy algorithm, and reactive control, respectively; b) Convergence of the stochastic algorithm to the optimal solution. Example with $N = 3$, $\varepsilon = 1/100$, $\theta_0 = 0$, $\theta_{\max} = 100$, and $p_0 = 30$.

Figure 5.6a depicts the comparison, for increasing number of applications, of the optimal solution, the `MDTA` algorithm, the benchmark greedy algorithm and the reactive control algorithm [109]. The reported results are averaged over 100 instances, with 95% confidence interval, for a scenario with parameters $n_i \in [1,5], \lambda_i \in [0.1, 0.3]$ s^{-1}, $\mu_i \in [10, 60]$ s^{-1} and $d_i \in [0.5, 3.3]$ s, under budget saturation.

Reactive control is a standard dynamic algorithm used in cloud for virtual machine (VM) migration: the algorithm performs online migrations in order to mitigate server overload conditions. When overload is detected, the VM with the highest ratio between the memory demand and the actual volume occupied by the VM is migrated. The actual migration is triggered when the server utilization goes beyond a certain threshold. This mechanism has been adapted to the fog placement problem by assuming that one application is placed in cloud every time the server utilization exceeds the threshold and, at the same time, the budget is sufficient to perform the offload. In the experiment of Figure 5.6a, the threshold is fixed as 40% of the fog server utilization. Every time a violation condition is detected, the application with the highest load in fog is chosen for the placement in cloud. The figure shows that a fixed-threshold strategy leads to large cumulative delay, especially under limited cloud budget. Also, since an optimal threshold policy accounts for the effect of processor sharing, as expected, it outperforms the greedy solution, which is based on the nominal load of the applications.

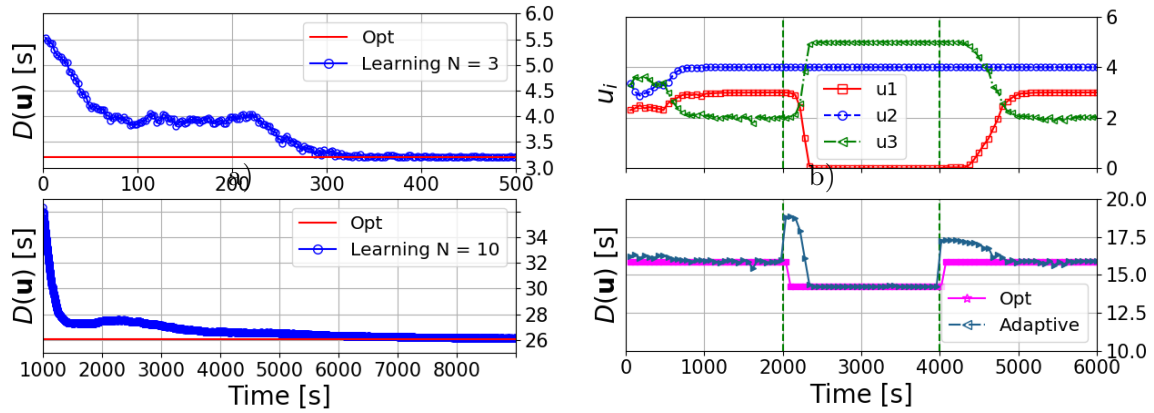

Figure 5.7: a) Convergence of the stochastic algorithm in terms of cumulative delay for $N = 3$ and $N = 10$; b) Dynamics of the three components and cumulative delay under the adaptive stochastic algorithm.

Figure 5.6b, Figure 5.7a, and Figure 5.7b evaluate the performance of the stochastic approximation algorithm. Figure 5.6b shows the convergence of the algorithm to the optimal solution for $N = 3$, where $\boldsymbol{\lambda} = (1.7, 1.2, 1.5)$ s^{-1}, $\boldsymbol{\mu} = (53.3, 108.9, 79.6)$ s^{-1}, $d_1 = 5$ s, $d_2 = 10$ s, $d_3 = 15$ s, $b_0 = 3$, $n_1 = 3$, $n_2 = 3$, and $n_3 = 1$. The convergence of the algorithm under an exponential and under a uniform distribution for the fog system

times is tested, using single samples as estimators. The algorithm reaches the optimal solution within 5000 iterations with a constant step-size $\epsilon_n = 1/100$, irrespective of the chosen distribution. Figure 5.7a reports the dynamics of the optimal cumulative delay under relatively slow batch arrival rates - 1.5 data batches per second per application on average - for $N = 3$ and $N = 10$, respectively; the algorithm converges to the optimal policy within 300 s and 6000 s, respectively.

Finally, Figure 5.7b describes the adaptive capabilities of the stochastic approximation algorithm. The graphic shows the dynamics of the components u_i and of the cumulative processing delay. For this example, the scenario has three applications with $n_1 = 3$, $n_2 = 4$, $n_3 = 5$ and two configurations: Config.1 = $\{\boldsymbol{\lambda}_1 = (3.4, 2.6, 3)\ \text{s}^{-1}$, $\boldsymbol{\mu}_1 = (103.3, 108.9, 79.6)\ \text{s}^{-1}\}$ and Config.2 = $\{\boldsymbol{\lambda}_2 = (1.4, 2.6, 3)\ \text{s}^{-1}$, $\boldsymbol{\mu}_2 = (5.33, 10.89, 7.96)\ \text{s}^{-1}\}$, respectively. At 2000 s, the batch arrival rate and the service rate of the first application suddenly drop, thus changing the optimal policy. The algorithm adapts to the configuration change, and converges to the optimal placement for the second configuration. After 4000 s the behaviour of the first application is restored to the initial configuration. Again, the algorithm reaches the optimal policy under Config.1. The learning algorithm proves able to detect sudden changes of the applications configuration in the dark, *i.e.,* without apriori knowledge of the system parameters.

5.9 Remarks and Possible Extensions

Fog applications may experience long processing delays on local servers due to processor sharing effects. In order to minimize the processing delay of a set of N fog applications, it is possible to execute the overloaded ones in cloud at a cost. In this context, a key metric is the marginal delay of fog applications measuring their relative performance when executed in cloud or in fog. Algorithms for the optimal placement have been introduced both under perfect information, with a search in the consumed budget, and when the load is unknown at runtime, a primal-dual stochastic approximation procedure can learn the optimal policy in the dark.

This work has only tackled a few aspects of the fog placement problem. First, while the proposed adaptive learning procedure works when stability conditions are met for all fog applications, a valuable objective is to perform in parallel also the online detection of the critically loaded ones, so as, *e.g.,* to proactively enforce their placement in cloud. Furthermore, the proposed framework can be extended to fog clusters accounting for, *e.g.,* standard servers placement policies adopted in cloud computing literature [113], where one needs to split the budget over specific servers, based on their current load.

We discuss some possible extensions of the work's model below.

5.9.1 Model Extension

Cloud Delay

One possible extension is to consider an additional cloud access delay, experienced by the applications having at least one module placed on the cloud. Positive cloud delay access prioritises the applications with lower number of modules n_i. Our model still holds valid by replacing (5.1) with

$$D_i(\mathbf{u}) = u_i\Big(G_i(u) - \frac{d_0}{n_i}\Big) + (n_i - u_i)\, d_i$$

where d_0 is the cloud access delay. The linear negative term added to $G_i(u)$ is a reward per module placed in fog: the access delay appears now in the marginal delay $r_i = G_i(u) - \frac{d_0}{n_i} - d_i$ and affects the order of modules of applications which are to be placed in fog.

With a fixed cloud delay, we have an additional fixed delay d_0 for each application that has at least one module deployed on the cloud. Hence, we add an indicator function (it indicates whether the application has at least one module deployed on cloud) to the delay definition of each application extending equation (5.1):

$$D_i(\mathbf{u}) = u_i G_i(u) + (n_i - u_i)\, d_i + d_0 \cdot \mathbb{1}_{\{u_i \neq n_i\}}. \tag{5.15}$$

The indicator function makes the objective function discontinuous. In order to use the concept of marginal delay we can distribute the cloud delay among all the applications and treat it like a reward per module placed in fog. In this manner we have new marginal delays defined as

$$r_i(u) = G_i(u) - \frac{d_0}{n_i} - d_i. \tag{5.16}$$

Such marginal delays retain a threshold structure of the solution.

Multi-server Scenario

The model is described for a single server for simplicity's sake. But, it extends to the case of M servers. Under a fixed dispatching policy, applications are assigned to fog servers, and for the ease of containers or VMs internetworking, it is current practice to have modules of the same application reside on same server. Hence, applications would be partitioned into M per-server groups and so the control variables. The cumulative delay becomes the sum of the cumulative delay per server, as a function of orthogonal per-server placement variables, coupled through the budget. The optimal orchestration policy has the same structure derived for the single server case, with marginal delays being now defined per server.

Multi-core Servers

In a multicore scenario VMs and containers will still perform depending on the server load and on the orchestrator's per-core dispatching policy. If such dispatching policy is known apriori, same rationale can be used as for the multiserver case. However, while the fog processing delay per application will increase at the increase of deployed modules, convexity may not necessarily hold; convex approximations of such delay characteristics should be used.

Chapter 6

Fog Orchestration meets Proactive Caching

Running fog computing applications on fog servers requires to match activation of applications containers to time varying demands. In this chapter we study the dynamic orchestration of a batch of applications over a network infrastructure including fog servers and a cloud. Cloud application deployment faces higher cost and high latency, but unlimited computational capacity. Fog servers, conversely, have limited computational resources, but ensure low latency at low cost. In this context we propose a new scheme for joint caching and placement in fog: the aim is to minimize the deployment cost while satisfying the applications' constraints. In fact, image caching appears mandatory to reduce the containers' activation time. On the other hand, proactively caching images on target servers is effective to match the expected activation pattern while optimizing load balancing via container replication. Using two-stage stochastic programming we derive a one-step-ahead policy to minimize the total running cost and satisfy applications' requirements. Extensive numerical results demonstrate the potential for this novel approach over traditional caching and placement algorithms. This chapter is mostly based on [51].

6.1 Introduction

Virtualization is key for the flexible installation of services onto fog servers in the proximity of IoT objects [23]. Indeed, heterogeneity of IoT technologies [75] is mitigated by packaging fog service modules in advance, *e.g.,* in the form of Docker *images* adapted to the host OS system [81]. Container-based orchestrators, such as Kubernetes [7], support availability and load balancing by means of container replication. Replicas of the image of a tagged fog application can be displaced on different target fog servers: once replicated among different servers, requests towards a tagged application are dispatched by applying a load balancing procedure (*e.g.,* Round-Robin) among the servers. On the other hand, for a fog server to offload some container, it has to query the controller for

the network topology in order to check which servers executing the same application can take over. The Kubernetes monitoring system ensures service availability by removing stalled containers and activating replicas on different servers.

Typically, containerized services can not be migrated in a stateful manner with a proper infrastructure. Rather, the practice is to replicate and switch on a container at the new location and turn off the old one. Clearly, this process introduces some delay which can be broken down into a *start up time*, a platform-dependent *orchestration overhead delay*, and the *image transfer delay*. The last component is critical because it involves unpredictable network delays and depends on the image size and on network availability. Thus, caching containers' images on fog servers is key to reduce the total delay experienced by fog applications. For instance, under Kubernetes, every node, as a Docker host, can perform image caching operations. A *cache hit* means that the container image is available on a target fog server and, if not active, the start-up time can be performed within some milliseconds [99]. A *cache miss*, conversely, requires either to wait the image download onto the fog server – a delay usually unacceptable for most applications – or to redirect the request on other available instances, *e.g.,* in cloud. Furthermore, every new image download involves several operations, such as the image decompression, with an overhead on the server's CPU usage much higher than image activation, especially on devices with limited resources [9].

In this chapter we study the joint optimization of application image caching and orchestration in fog. We consider single container fog applications, and we leave the extension to more complex microservice architectures for later works. The system infrastructure includes a central cloud and a fog region with a cluster of fog servers. Fog servers have limited CPU, memory and storage capacity but are cost free. In cloud, unlimited computational resources are available at a cost. The aim is to optimally orchestrate applications running on the described infrastructure. I.e., minimize deployment costs while complying with applications' delay figures.

Inspired by proactive caching systems, we study an optimal joint orchestration and caching policy. Proactive caching is aimed at preventing the download of containers from a central repository to avoid large image transfer delay. The control is the proactive placement of the applications' images stored in the container registries across the network infrastructure, mapping applications' containers to fog servers or to the cloud. Each application's container can be either cached or not on a fog server. Also, it can be available in two forms on each server: it can be either *active*, *i.e.,* the container is running on the server, or it can be *disabled*, *i.e.,* the image of the container is cached but not yet running.

State-of-the-art solutions for images caching, such as Kubernetes' caching for instance, are *reactive* and thus not suitable for delay-constrained fog computing applications. In fact, a container not present on a target host is fetched from the central repository (*e.g.,* a Docker hub). Proactive edge caching has become popular in 5G networks [18] and has

Table 6.1: Main notation used throughout the chapter

Symbol	*Meaning*
N	number of applications
S	number of fog servers
λ_i^t	arrival rate at time t for application i
δ_i	maximum processing delay tolerable by application i
r	resource type: processing (P), memory (M) and storage (D).
C_s^r	available resource in server s, with $r \in \{P, M, D\}$
c_i^r	resource occupation of application i container, $r \in \{P, M, D\}$

appeared recently in the literature [58].

However, fog proactive caching appears fundamentally different compared to standard multimedia edge caching. First, it maybe tempting leveraging on standard caching policies as done in proactive multimedia edge caching. Most frequently used (MFU), for instance, is provably optimal under stationary content popularity [76]. In fog computing such analogy is misleading because the more requests a container receives, the more the processing resources consumed in terms of CPU, storage and communications on the servers it is installed on. Thus, the server "occupation" depends on the fog application's "popularity" and a fog application's performance depends by the presence of other active containers on the same host. As proved by our numerical experiments, traditional caching techniques are suboptimal for the considered problem, especially under variable demand patterns.

Main contribution: we propose a one-step-ahead joint caching and orchestration scheme accounting simultaneously for caching, load-balancing, and replication of fog containers' images. One application may be cached in advance on several servers and may be activated or not depending on the current application's load. In our model, the CPU load generated at a specific server depends on the number of replicas installed elsewhere and made run in parallel precisely to smooth the peak processing delay per instance. Our work is, to the best of the authors' knowledge, the first one to study jointly proactive containers' image caching and orchestration in a fog computing scenario.

6.2 System Model

We consider a network infrastructure consisting of a fog cluster and a cloud. The fog cluster is composed by S *fog servers*. Each server s has a limited amount of CPU, memory and disk storage available, denoted by the triple $\mathbf{C_s} = (C_s^P, C_s^M, C_s^D)$. On the other hand, in cloud there are unlimited but expensive resources.

Applications. We consider a batch of N different applications to be run on the infrastructure; they can be deployed either in cloud or on the fog cluster. Each application consists of a container to be replicated across several fog servers and/or in cloud.

Each container of application i has requirements in terms of CPU, memory and storage, described by the triple (c_i^P, c_i^M, c_i^D).

Cache. We assume that each fog server has the ability to store containers in a *cache*, and possibly to disable cached containers when not needed. This allows to predict future application requests and reduce start-up delays for new applications, that is to download the respective containers if not cached locally. We consider that each container may consist of successive *layers*, corresponding to incremental software updates. Yet, out of all layers downloadable from a central repository, a container of application i only needs a portion $\eta_i \in (0, 1]$ of them to run properly.

Control. The orchestrator decides at each period t:

1. Which new containers should be downloaded to the fog servers' caches;
2. Which containers, among those who are are present in the respective caches, should be activated.

We remark that the activation policy is constrained by the delay requirements of each application, and we shall provide some constraints on the churn rate of cached images in order to disincentive disruptive reconfigurations.

Let $x_{i,s}^t$ be the binary variable indicating whether a container of the application i is active on server s at time t ($s = 0$ denotes the cloud for the notation's sake). The caching control is represented by continuous variable $y_{i,s}^t \in [0, 1]$: it describes the fraction of container's layers cached on server s at time t for application i. In fact, containerized applications may be operational even when some features are missing. But, if $y_{i,s}^t < \eta_i$, *i.e.,* the portion of cached layers is insufficient, then $x_{i,s}^t = 0$, hence the application can not be activated. We denote

$$\mathcal{A}(y^t) = \{(i, s) | y_{i,s}^t < \eta_i\}$$

the set of containers that can not be activated: $x_{i,s}^t = 0$, for all pairs $(i, s) \in \mathcal{A}(y^t)$.

Requests. For the sake of model tractability we divide the time into slots. During slot t, application i receives data to be processed at a rate of λ_i^t – that can be thought of as a measure of application's *popularity* – and we denote by $\bar{\lambda}^t$ the vector of all arrival (or demand) rates. Although theoretically a time-slot can be defined at one's will, in practice we suggest to define a new slot whenever the arrival rates change considerably. This assumption is aligned with the majority of monitoring systems. Indeed, in most of the existing systems as Kubernetes [7], a new orchestrating decision is taken whenever a change in the applications arrival rates is detected and a potential inconsistency between the required and guaranteed requirements is detected. We remark that demand rates λ at future slots $t' > t$ are not know, but can only be predicted; a discussion on models for IoT data traffic is beyond the scope of this work [82].

Latency. The presence of several active applications on the same fog servers impacts the processing time of each of them. The processing delay $d_{i,s}^t$ experienced by a tagged

application i on a server s at time t can be modelled as a convex function of the application's demand rate, and of the the number of applications active on the server and processing capacity of the server:

$$d_{i,s}^t(\bar{x}_s^t, \lambda_i^t) = \left(\frac{c_i^P}{\sum_{i=1}^N x_{i,s}^t} - \frac{\lambda_i^t}{\sum_{s=0}^S x_{i,s}^t} \right)^{-1} \tag{6.1}$$

By (6.1) over-exploiting fog servers by increasing container replicas reduces the CPU share each receives thus increasing the processing delay. Conversely, a uniform load balancing policy can split the demand rate per application evenly across all servers on which the application is active. On the other hand, in cloud there is no computational bottleneck, so that a container of application i activated in cloud has constant processing time d_0^i. However, we assume a fixed latency Δ_0 between the fog cluster and cloud.

Cache churn rate. The cache storage occupation can not exceed the cache capacity C_s^D, *i.e.,* $\sum_{i=1}^N y_{i,s}^t \, c_i^D \leq C_s^D$, for each fog server s. Also, due to limited download speed, we fix an upper bound ϵ on how much cache content can vary between consecutive slots: a fair resource share imposes

$$|y_{i,s}^{t+1} - y_{i,s}^t| \leq \epsilon,$$

for each application i and server s.

Orchestration constraints. We assume application i to tolerate a maximum processing delay of δ_i seconds. When active in fog, this is described by constraint

$$d_{i,s}^t(\bar{x}_s^t, \lambda_i^t) \, x_{i,s}^t \leq \delta_i,$$

for any fog server s. In cloud, the fog-to-cloud latency is factored in as

$$(d_0^i + \Delta_0) \, x_{i,0}^t \leq \delta_i.$$

Furthermore, CPU and memory occupation of all containers activated on a given server s can not exceed the total CPU and memory available at time slot t, *i.e.,*

$$\sum_{i=1}^N x_{i,s}^t \, c_i^r \leq C_s^r,$$

for every request $r \in \{P, M\}$.

Finally, every application i for which $\lambda_i^t > 0$ must be served at time slot t, so that at least one active container is needed, *i.e.,*

$$\sum_{s=0}^S x_{i,s}^t \geq 1.$$

Objective. In order to minimize the financial cost of running the overall infrastructure, we aim at minimizing the number of containers $\sum_{i=1}^N x_{i,0}^t$ deployed in cloud at each time-slot t. We do so by jointly controlling caching and activation of containers, while fulfilling caching and orchestration constraints, per server and per application as described above.

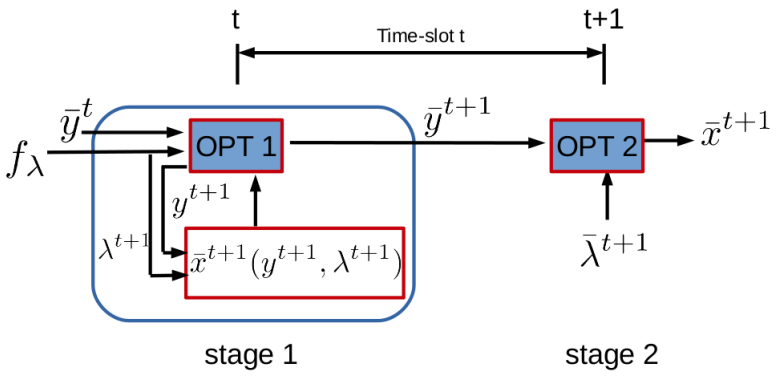

Figure 6.1: Two-stage optimization and its one-step ahead (OSA) solution

6.3 Solution: One-step ahead programming

Our goal is to devise an orchestration policy that jointly decides the caching and the activation of containers for the requested applications with the aim of minimizing the number of containers deployed in the cloud. This allows to eventually minimize the overall deployment cost.

The frequency at which new requests arrive and (already cached) containers are activated is much higher than the frequency at which new containers can be downloaded and cached at fog servers [99]. For this reason, we decide to articulate the decisions on caching and activation in hierarchical and sequential fashion. Caching decision are planned in advance in stage 1, to anticipate the scenario that may materialize in stage 2 once the container download is terminated. Finally, at stage 2, new containers are cached in the servers and they are activated in accordance with the actual requests.

We remark that the two stages are interleaved in time: at time t, the caching decisions are made (stage 1), while containers are activated given the caching decisions taken at previous time $t-1$ and materialized at time t (stage 2).

Technically speaking, our solution approach follows the classic paradigm of *two-stage stochastic programming* [20] and its natural *one-step ahead* (OSA) associated solution, illustrated in Figure 6.1 and described formally below. We will describe the two stages in backward fashion, since stage 1 relies on the solution of the problem solved by stage 2.

6.3.1 OSA - Stage 2: Activation of cached containers

By time $t+1$, the new arrival rates $\bar{\lambda}^{t+1}$ are observed. Moreover, we assume that stage 1 has already taken place at time step t, producing caching decision $\bar{y}^{t+1}$; hence, by time $t+1$ new containers are downloaded accordingly. Then, as the second stage of our optimization, cached containers are activated so as to minimize the number of them in cloud, while fulfilling all the orchestration constraints. The corresponding mathematical

program writes as follows.
`Input`: Actual arrival rates $\bar{\lambda}^{t+1}$, container caching $\bar{y}^{t+1}$
`Output`: Container activation $\bar{x}^{t+1}(\bar{y}^{t+1}, \bar{\lambda}^{t+1})$
`Compute` at time $t+1$:

$$\bar{x}^{t+1}(\bar{y}^{t+1}, \bar{\lambda}^{t+1}) = \underset{x^{t+1}}{\operatorname{argmin}} \quad \sum_{i=1}^{N} x_{i,0}^{t+1} := F^{(2)}(x^{t+1}) \tag{OPT2}$$

subject to:

$$\begin{aligned}
& d_{i,s}^{t+1}(x_s^{t+1}, \bar{\lambda}_{i,s}^{t+1})\, x_{i,s}^{t+1} \leq \delta_i, \\
& \forall i \in \{1,\dots,N\}, \forall s \in \{1,\dots,S\} \\
& (d_0^i + \Delta_0)\, x_{i,0}^{t+1} \leq \delta_i, \quad \forall i \in \{1,\dots,N\} \\
& x_{i,s}^{t+1} = 0, \quad \forall (i,s) \in \mathcal{A}(\bar{y}^{t+1}) \\
& \sum_{s=0}^{S} x_{i,s}^{t+1} \geq 1, \quad \forall i : \bar{\lambda}_i^{t+1} > 0 \\
& \sum_{i=1}^{N} x_{i,s}^{t}\, c_i^r \leq C_s^r, \quad \forall s \in \{1,\dots,S\}, \forall r \in \{P, M\} \\
& x_{i,s}^{t+1} \in \{0,1\}, \quad \forall i \in \{1,\dots,N\}, \forall s \in \{1,\dots,S\}.
\end{aligned} \tag{6.2}$$

6.3.2 OSA - Stage 1: Container caching

At time t, stage 1 has to devise a *caching policy* $\bar{y}^{t+1}$ deciding the percentage[1] of container layers to be cached in each server. Future arrival rates at time $t+1$ are unknown and can only be predicted, hence $\bar{y}^{t+1}$ is computed by minimizing the *expected* number of containers deployed on the cloud at time $t+1$. Yet, for each *possible* arrival rate scenario λ^{t+1} and for each caching $\bar{y}^{t+1}$ one can compute the optimal container activation $\bar{x}^{t+1}(\bar{y}^{t+1}, \lambda^{t+1})$ through the analogous version of (OPT2).

Hence, we define the objective function $F^{(1)}$ of the first stage optimization as the expected value of the objective realized of stage 2, *i.e.,* $\mathbb{E}_{f_\lambda}[F^{(2)}]$. Here, f_λ is the predicted arrival rate distribution at time $t+1$, *i.e.,* $f_\lambda(a) = \Pr(\lambda^{t+1} = a)$. We can also account for the fact that more up-to-date containers will generally provide better performance by introducing an increasing function $R(.)$ of the number of cached container layers, which results in the following objective function

$$F^{(1)}(y^{t+1}, \bar{x}^{t+1}) := \mathbb{E}_{f_\lambda}\left[F^{(2)}(\bar{x}^{t+1})\right] - \sum_{i,s} R(y_{i,s}^{t+1}) \tag{6.3}$$

where $\bar{x}^{t+1} := \bar{x}^{t+1}(\bar{y}^{t+1}, \lambda^{t+1})$. The optimization problem solved at stage 1 is described below.

[1] relative to the most recent container version present in the main repository

`Input`: Predicted arrival rates distribution f_λ for time $t+1$

`Output`: Caching policy $\bar{y}^{t+1}$

`Compute` at time t:

$$\bar{y}^{t+1} = \underset{y^{t+1}}{\operatorname{argmin}} \quad F^{(1)}(y^{t+1}, \bar{x}^{t+1}(y^{t+1}, \lambda^{t+1})) \tag{OPT1}$$

subject to:

$$\sum_{i=1}^{N} y_{i,s}^{t}\, c_i^D \leq C_s^D, \quad \forall s \in \{1, \ldots, S\} \tag{6.4}$$

$$|y_{i,s}^{t+1} - y_{i,s}^{t}| \leq \epsilon, \quad \forall i \in \{1, \ldots, N\}, \quad \forall s \in \{1, \ldots, S\} \tag{6.5}$$

$$y_{i,s}^{t+1} \in [0,1], \quad \forall i \in \{1, \ldots, N\}, \forall s \in \{1, \ldots, S\}$$

We highlight that at this stage one optimizes over both container caching y^{t+1} and activation $\bar{x}^{t+1}(y^{t+1}, \lambda^{t+1})$ for *each* possible scenario λ^{t+1}. However, *only* caching decisions $\bar{y}^{t+1}$ are deployed in the system. In fact, container activation for time $t+1$ is performed only once the true requests $\bar{\lambda}^{t+1}$ materialize at time $t+1$, since activation is almost instantaneous in practice. Hence, at stage 1 the activation variables x are only auxiliary, as they have the sole purpose of evaluating the quality of caching.

6.4 Solving Stage 1: Derivative-free methods

As described above, the caching problem (OPT1) requires to solve (OPT2) for each possible demand rate scenario λ^{t+1}. This entails two major technical difficulties:

1. (OPT2) is computationally hard (being formulated as an Integer Program);
2. The objective function $F^{(2)}$ is not in closed-form.

Regarding *1)*, we describe below a simple heuristic for (OPT2). To tackle *2)* we resort to *derivative-free* optimization techniques, that only need to sample the value of the function, without needing to resort to their derivative. The general paradigm we employ is coordinate-descent [89], which at each iteration selects one coordinate $y_{i,s}$, keeps the others fixed, and optimizes function $F^{(1)}$ over $y_{i,s}$ via a univariate, derivative-free line search method such as Bayesian Optimization (BO) [107] or Golden Section Search (GSS) [70].

We dub this procedure Coordinate-Descent-Caching (CDC) and we describe it in Algorithm 4. There, K is number of iterations; *coord_ select* and *search* are the procedures for the coordinate variable selection and the derivative-free line search method, respectively. Each time a new coordinate (i, s) is selected, a lower and an upper bound for $y_{i,s}$ are computed in lines 4 and 5 to confine the search, respectively. Specifically, the upper bound u for variable $y_{i,s}$ is the maximum between the value for the *cache churn rate* and the residual storage capacity of server s with all $y_{j,s}$s, $j \neq i$, fixed. This allows CDC to output a feasible solution.

Algorithm 4: `Coordinate-Descent-Caching(CDC)`

Input: arrival distribution f_λ, $\bar{y}^t$, η, $\epsilon > 0$
Output: Caching decisions $\bar{y}^{t+1}$

$\bar{y} \leftarrow \bar{y}^t$;
for $k = 1, \ldots, K$ **do**
  $(i, s) \leftarrow coord\_select(N, S)$;
  $l \leftarrow \max\{y^t_{i,s} - \epsilon, 0\}$;
  $u \leftarrow \min\{y^t_{i,s} + \epsilon, \frac{C^S_s - \sum_{j \neq i} c^S_j \, y_{j,s}}{c^S_i}\}$;
  $y_{i,s} \leftarrow search(F^{(1)}(\bar{y}, \bar{x}^{t+1}(\bar{y}, \lambda^{t+1})), f_\lambda, [l, u], \bar{\eta})$;
$\bar{y}^{t+1} \leftarrow \bar{y}$;
return $\bar{y}^{t+1}$

Proposition 6.1. *At each iteration of CDC it holds that*

$$\bar{y}^{t+1} \in \mathcal{Y}^{t+1} := \left\{ y^{t+1} \in [0,1]^{N \times S} | (6.4),\ (6.5)\ hold \right\}. \tag{6.6}$$

Hence, the caching solution computed by CDC is feasible for (OPT1).

The proof of Proposition 6.1 trivially follows from the computation of bounds l and b in Algorithm 4.

6.4.1 Heuristic for (OPT2)

As mentioned, optimizing stage 1 requires to solve the container activation problem (OPT2) as a sub-routine, for each possible demand rate λ^{t+1}. This clearly calls for a heuristic approach to solve (OPT2)) which is originally formulated as an Integer Program. We propose a greedy placement algorithm which selects, for each application i, an admissible set of fog servers where a container can be activated. Hence, the server with minimum memory and CPU occupation is chosen. Once this applications-servers mapping is obtained, if all the applications delay constraints are met then the mapping is considered as a valid activation. Otherwise, the applications violating constraint are moved to the cloud if their fog-to-cloud latency permits. We note that (OPT2) performs also load balancing by containers replication among the fog servers whereas our heuristic does not; this will be studied in future work.

6.4.2 Coordinate Selection and Search methods

The *coord_select* procedure can have several variants; we choose a method where coordinates are randomly permuted and selected sequentially. Other methods can be envisioned, *e.g.,* coordinate selection on the basis of the real arrival rates of the previous time-slot t. In this way, applications with highest arrival rates in the previous time-slot would be prioritized for the caching in the fog servers. We leave such variants for future works.

Table 6.2: Applications' containers requirements [27].

Requirement	*Mean Value*	*Range*
CPU	1250 MIPS	$[500, 2000]$ MIPS
Memory	1.2 Gbytes	$[0.5, 2]$ Gbytes
Storage	3.5 Gbytes	$[1, 8]$ Gbytes

For the derivative-free line *search* procedure we evaluated Bayesian Optimization (BO) [107] and Golden Section Search (GSS) [70]. GSS is a classic dichotomy procedure that samples the function at two middle points of the current search interval and then restricts the interval. It returns the global optimum of a univariate unimodal function, otherwise – which is our case – it returns a local optimum. The BO method is usually applied when the utility function is expensive to evaluate because it has high sample efficiency. Indeed, the term related to the caching computation $(x_{i,0}^{t+1}(\bar{y}^{t+1}, \bar{\lambda}))$ is inherently computationally expensive to evaluate, even by using a heuristic activation as we do. By inferring the function at unknown points via a Gaussian Process, BO selects points having high probability of achieving low cost.

6.4.3 Computational Complexity

The computational complexity of CDC is mainly dominated by the computation of the placement function $\bar{x}^{t+1}(y^{t+1}, \lambda^{t+1})$, the size $|f_\lambda|$ of the support of f_λ (determining the number of possible demand rate scenarios λ^{t+1}), and the search method. The computational complexity of the greedy heuristic for container activation is $O(NS)$. Under sequential coordinate selection and with a fixed number of iterations K, the total complexity is $O(|f_\lambda|KNS\log(\tau^{-1}))$. The logarithmic factor appears due to the convergence rate of iterative methods such as the Golden Section Search method [70] where τ is a tolerance parameter. Hence, remarkably, the total complexity remains polynomial in the size of the input.

6.5 Numerical Results

In this section we evaluate our solution in a specific fog computing scenario where we test our joint caching and orchestration scheme. Three main goals are in order:

1. Select the best method to perform the proactive caching optimization, *i.e.,* select the most suitable coordinate descent algorithm using two candidate search methods, namely Golden Section Search (GSS) and Bayesian Optimization (BO).

2. Compare our approach with standard placement algorithms used to drive the activation step.

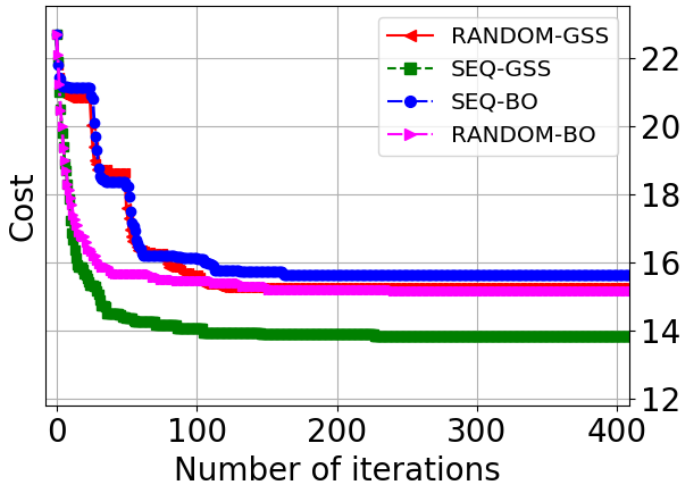

Figure 6.2: Expected cost of GSS and BO derivative-free line-search methods for $N = 25$, averaged across 10 instances.

3. Demonstrate that our approach outperforms baseline schemes in terms of cloud deployment cost.

6.5.1 Simulation Settings

In our simulation experiments we consider one input batch of applications to be deployed on a system composed of one fog region with 3 servers and a cloud. Resources and applications requests per server are represented by triples of *CPU*, *memory* and *storage* units. Servers' capacity triples are $\mathbf{C_1} = (15000, 8, 50)$ and $\mathbf{C_2} = \mathbf{C_3} = (44000, 16, 60)$, where CPU is measured in MIPS and memory in Gbytes. The applications' requirements per container are listed in Table 6.2. In our tests, we have assumed different sizes of the applications' batch, namely $N = 10, 15, 20, 25$. The demand rates and the maximum tolerable processing delay of each application are generated uniformly at random in $[0, 300]$ jobs/s and in $[5, 6]$ sec, respectively. Fog-to-cloud latency has been set to 500 ms, and the cloud processing delay is generated uniformly at random in $[1, 5]$ sec, for each application.

6.5.2 Search Methods

In stage 1, the orchestrator computes the caching policy by the coordinate descent Algorithm 4 (CDC) described in Section 6.4. As a sub-routine, CDC computes the minimum of the objective function $F^{(1)}$ on a line via a derivative-free method. We hence compared Golden Section Search (GSS) and Bayesian Optimization (BO) on Gaussian Processes. Figure 6.2 shows the expected cost of deployment achieved by the two methods for $N = 25$.

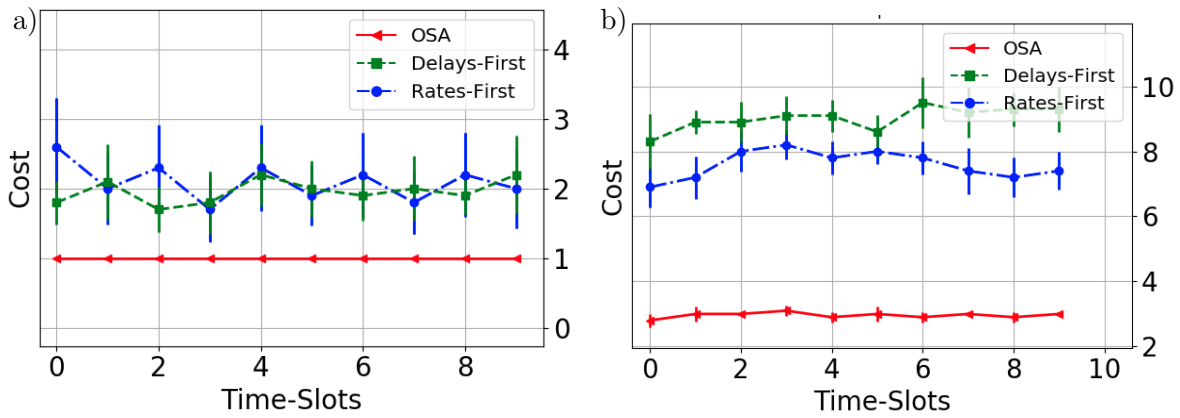

Figure 6.3: a) Evaluation of placement methods for $N = 10$; b) Evaluation of placement methods for $N = 15$.

Each point is the average of ten instances where the infrastructure is fixed and the application arrival rates' distribution changes. We implemented two variants for the coordinate selection method. The *sequential* (SEQ) approach applies each method to each coordinate of the caching matrix $\bar{y}^t$ in sequential manner. The *random* (RANDOM) approach selects a random coordinate at each iteration. The figure highlights the capability of GSS to obtain a better value of the expected cost with respect to the BO method. BO is sensitive to input hyper-parameters, and it is generally no easy task to select the most suitable ones. Given its better performance, in the next experiments we adopt GSS as our preferred search algorithm as a sub-routine in CDC, to solve stage 1.

6.5.3 Container activation algorithms for Stage 2

We now discuss how to solve the container activation problem (OPT2) in stage 2. We devised two reasonable baseline heuristics for the joint caching and orchestration problem. We call *Delays-first* and *Rates-first* our baseline activation algorithms: they give priority to applications with the lowest delay constraints and highest demand rates, respectively. These two heuristics apply both to the activation function $x_{i,0}^{t+1}(\bar{y}^{t+1}, \bar{\lambda})$ used in the first stage of our approach, *i.e.,* to cache containers, and to the activation phase of the second stage; yet, the baseline heuristics do not perform proactive container replication. The caching policies are standard proactive edge caching ones as described later on.

Figure 6.3 shows the overall performance comparison among the proposed methods, averaged over 10 instances along with their 95% confidence interval. We set $\epsilon = 0.6$ for the *cache churn rate* constraint, meaning that at most 60% of cached containers can be changed at each step. We can observe from the figure that heuristics are far from the optimal one, which in turn is attained by our approach (OSA). In fact, prioritizing containers' activation on the basis of their delay constraints would lead to deploying appli-

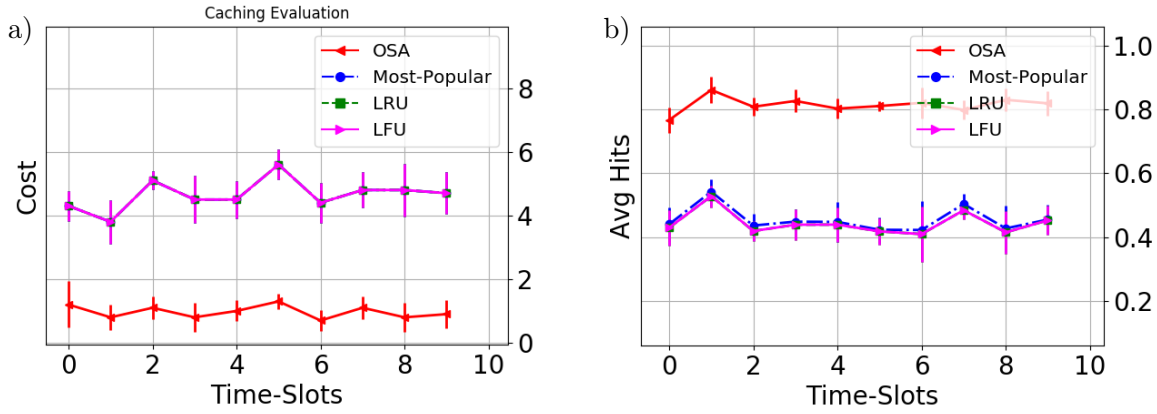

Figure 6.4: a) Cost incurred by each caching policy; b) Average number of hits per fog server.

cations with strict constraints in fog; this will cause an increase of the delay experienced per application due to CPU sharing on fog servers. Lack of replication of same containers on different servers increases the delay as well. Thus, all the remaining applications will be deployed in cloud, hence incurring a larger cost. A similar argument applies to the *Rates-first* heuristics since in that case applications with higher demand rates will be deployed together on the fog servers; but this is possible as long as their delay constraints are satisfied, while other applications with lower arrival rates must be deployed in cloud.

6.5.4 Caching

The performance gain of our one-step-ahead (OSA) approach over the two heuristics can be ascribed to its efficient caching strategy. Hence, we now highlight the key differences between standard proactive edge caching and our optimized proactive fog caching. We consider the following classical caching strategies for comparison:

- the *Most-Popular (MP)* which, at each time-slot t, for each server, prioritizes applications with highest popularity, *i.e.*, applications presenting the highest arrival rate at the previous time-slot t.
- *Least Recently Used* (LRU) strategy, evicting the least recently requested containers on each server.
- *Least Frequently Used* (LFU) strategy which discards the least frequently requested container since the first time-slot.

MP, LRU and LFU strategies track demand rates and demand epochs by updating a table of file requests.

We imposed the *cache churn rate* constraint, with $\epsilon = 0.3$, for all caching strategies. Also, each file is cached at the minimum level required for its activation. LRU, LFU

and MP refer to the plain memory occupation as the reference metric to evaluate cache space on fog servers. Figure 6.4 reports on the performance of each fog caching policy for $N = 20$. Data points represent an average over ten instances. For each instance a distribution over the applications' demand rates is generated and, at each time-slot, a vector of arrival rates is chosen with probability defined by the initial distribution. In these experiments all the vectors of demand rates are sampled with uniform probability. In Figure 6.4a) the comparison in terms of deployment cost is showed. All classical caching policies have same cost due to the low percentage of discarded files during the sampled period, whilst approach (OSA) performs significant savings. Furthermore, Figure 6.4b) shows the average number of hits per fog server. The highest rate is achieved by OSA with respect to baseline approaches, as expected. These results prove that in a fog environment joint caching and orchestration of applications is key in order to reduce expensive costs of deployment in cloud.

6.6 Remarks and Possible Extensions

In this chapter we have developed a framework to perform the orchestration of fog applications and minimize their deployment cost. We have proved that proactive caching greatly improves the performance of fog orchestration by matching in advance the expected demand rates of applications and the available resources on fog servers. Actually, fundamental differences exist between fog caching and edge caching due to the multidimensionality of resources per deployed container, the impact of fog-to-cloud delay and the effect of resource sharing on fog servers. Our scheme performs a two-stage stochastic optimization by forecasting the impact of containers' activation onto servers capacity and applications computing delays.

Several novel aspects deserve further study to this respect. Some possible extensions are highlighted in the following points:

- *Greedy Heuristic for OPT2.* One possible extension involves the formulation of a new lightweight heuristic solution to approximate the behaviour of the optimal one for OPT2. Indeed, the actual adopted solution is a very basic one. A possible improved version could consider the possibility of replication in each available server for each application's containers. Replication, indeed, is beneficial to lower the processing delay experienced by the single applications. However, the heuristic should be able to find the right trade-off between applications' replication degree and the servers occupation level, another crucial element that greatly influences the processing delay of applications.

- *Alternative formulation for OPT2.* One fundamental constraint of OPT2 is that each container's deployment must satisfy the delay bound of the application. However, in real scenarios, delay violations are quite likely implying an infeasibility of the

problem. For this reason, the formulation of OPT2 can be relaxed admitting the violation of delay constraints in case of cloud deployment. In this way, a penalty function could be added in order to minimize the number of such violations improving the feasibility of the model.

- *Generalization of delay function.* We adopted a specific model for the processing delay function. A possible extension is represented by the generalization of such a function showing the validity of the model even with other delay functions.

- *Adaptive algorithmic solutions.* An important open question is how to dynamically adjust the solutions found by the aforementioned algorithms as demands and resources vary over time.

- *More scalable solutions.* Another interesting extension consists of investigating other scalable solutions as the number of involved variables increases. Indeed, depending on the length of a time-slot, *i.e.,* the frequency at which new requests arrive, different derivative-free optimization methods can be evaluated. The coordinate-descent method proposed could suffer from scalability problems as the arrival rates frequency increase. Hence, a deeper study of alternative iterative optimization methods such as, *e.g., Sequential Quadratic Programming*, can represent a significant improvement for the current work.

Chapter 7

Conclusions

The declared aim of the thesis in the introduction chapter was "to contribute to the study and the design of new resource allocation algorithms in distributed and heterogeneous computing systems such as fog computing". In order to achieve such an objective, we detected two main challenges for resource allocation algorithms tackling the applications deployment problem in this context.

The first one is the placement problem where, given a batch of microservice-based applications with complete information of their requirements, and a distributed region-based fog infrastructure, the objective is to deploy the applications on the infrastructure optimizing the costs of the infrastructure's owner.

To this scope, in Chapter 3 we introduced a joint partitioning and optimization framework for the deployment of throughput-intensive applications on a region-based fog infrastructure. Experimental results showed that a smart cut of the applications' graph is beneficial both for improving the applications' performance requirements and to maximize the revenue of the infrastructure's owner. Given the lack of solutions for region-based deployments of microservice applications, this work improved the existing cloud-native solutions, proposing a new framework that fosters the inter-operability between fog regions.

In Chapter 4 we considered the problem of applications placement in a federated cloud-fog environment. Here, we showed that a breadth-first exploration of the solution space presents better performance with respect to standard solutions, mostly when applications present locality constraints requiring the placement of microservices on regions of the main domain. Furthermore, we highlighted the importance of trade-off feasibility, cost-efficiency and scalability in such a distributed and heterogeneous environment.

The second challenge we detected for resource allocation algorithms in this context, is the possibility to optimize the QoS performance of applications when the arrival requests are not completely known at runtime, *i.e.,* the dynamic orchestration problem.

From this aspect, Chapter 5 tackled the problem of orchestration of applications pipelines between a local fog server and an expensive cloud. We modelled each application as a cascade of queues and we considered a processor sharing model for the local

server. With such a model we tried to capture the bottleneck, in terms of performance degradation, that each application experiences when it shares computational resources with other applications deployed on the same server. This represents a novelty in the fog literature, especially for applications consisting of a set of interdependent modules deployed in resource constrained nodes. We provided an optimal algorithmic solution under perfect information about the arrival rates of each application and, finally, we formulated a stochastic approximation scheme in the case the load and the arrivals rates are not known at runtime. The proposed solution present relevant convergence and adaptability properties.

Finally, in Chapter 6 we introduced a proactive caching system that helps applications to avoid long start-up times that degrade their performance. We proposed an optimisation scheme that jointly takes into account the caching of application on fog servers and the strategic activation of that applications with unknown arrival requests for each applications. The activation of an application on a given server influences the performance of all other applications currently active on the server. This sheds new light on the caching problem in the context of fog computing. The proposed proactive caching system greatly improves the fog orchestration by matching in advance the expected demand rates of applications and the available resources on fog servers.

Overall, we believe that the research work of this thesis contributed to extend the state of the art by characterising new research lines straddling networking and computing problems. A consistent part of the results of this thesis can be promptly used in current fog computing platforms that adopt state-of-the-art technologies. However, further work is needed to explore and exploit the full potential of such a new computing paradigm.

References

[1] IoT and Predictive Analytics: Fog and Edge Computing for Industries versus Cloud https://shorturl.at/boG15.

[2] Amazon IoT Greengrass https://aws.amazon.com/greengrass/.

[3] Microsoft Azure IoT Edge https://azure.microsoft.com/en-us/services/iot-edge/.

[4] Industrial Internet Consortium. http://www.iiconsortium.org/.

[5] Kubernetes scheduler. https://github.com/kubernetes/.

[6] KubeEdge: an open platform to enable Edge computing. https://kubeedge.io/.

[7] Kubernetes. http://kubernetes.io/.

[8] FogAtlas. https://fogatlas.fbk.eu/.

[9] Arif Ahmed and Guillaume Pierre. Docker-pi: Docker container deployment in fog computing infrastructures. *International Journal of Cloud Computing*, 9(1):6–27, 2020.

[10] Md Mostofa Akbar, M Sohel Rahman, Mohammad Kaykobad, Eric G Manning, and Gholamali C Shoja. Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Computers & operations research*, 33(5):1259–1273, 2006.

[11] Eitan Altman. *Constrained Markov Decision Processes*. Chapman & Hall, '99.

[12] Eitan Altman, Konstantin Avrachenkov, and Urtzi Ayesta. A survey on discriminatory processor sharing. *Queueing Systems*, 53(1):53–63, Jun 2006.

[13] Eitan Altman and Nahum Shimkin. Individual Equilibrium and Learning in Processor Sharing Systems. *Operations Research*, 46(6):776–784, December 1998.

[14] Edoardo Amaldi, Stefano Coniglio, Arie MCA Koster, and Martin Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, 2016.

[15] Jean-Paul Arcangeli, Raja Boujbel, and Sébastien Leriche. Automatic deployment of distributed software systems: Definitions and state of the art. *Journal of Systems and Software*, 103:198–218, 2015.

[16] Ram Govinda Aryal and Jorn Altmann. Dynamic Application Deployment in Federations of Clouds and Edge Resources using a Multiobjective Optimization AI Algorithm. In *IEEE International Conference on Fog and Mobile Edge Computing (FMEC)*, 2018.

[17] Rami Atar and Mark Shifrin. An asymptotic optimality result for the multiclass queue with finite buffers in heavy traffic. *Stochastic Systems*, 4(2):556–603, 2015.

[18] E. Bastug, M. Bennis, and M. Debbah. Living on the edge: The role of proactive caching in 5G wireless networks. *IEEE Communications Magazine*, 52(8):82–89, Aug 2014.

[19] D. Bertsekas. Convexification procedures and decomposition methods for non-convex optimisation problems. *Journal of Optimisation Theory and Applications*, 29(2):169–197, 1979.

[20] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[21] Luiz F Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F Rana, and Manish Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, 2017.

[22] Luiz Fernando Bittencourt, Márcio Moraes Lopes, Ioan Petri, and Omer F Rana. Towards virtual machine migration in fog computing. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015 10th International Conference on*, pages 1–8. IEEE, 2015.

[23] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[24] Sem Borst, Rudesindo Nunez Queija, and Bert Zwart. Sojourn time asymptotics in processor-sharing queues. *Queueing Syst.*, 53:31–51, 06 2006.

[25] Stephen Boyd and Lieven Vandenberghe. *Convex Optimisation*. Cambridge University Press, 2004.

[26] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. How to Best Deploy Your Fog Applications, Probably. In *IEEE International Conference on Fog and Edge Computing (ICFEC)*, 2017.

[27] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. How to best deploy your Fog applications, probably. In *Proc. of IEEE ICFEC*, pages 105–114, 2017.

[28] P.J. Burke. The output of a queuing system. *Operations Research*, 6(4):699 – 704, 1956.

[29] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Comm. of the ACM*, 59(5):1837–1852, May 2016.

[30] R. Buyya, R. Ranjan, and R. N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *2009 International Conference on High Performance Computing Simulation*, pages 1–11, 2009.

[31] L. Canzian and M. V. Der Schaar. Real-time stream mining: online knowledge extraction using classifier networks. *IEEE Network*, 29(5):10–16, Sept. 2015.

[32] H. Cao, H. Hu, Z. Qu, et al. Heuristic Solutions of Virtual Network Embedding: A Survey. *China Communications*, 15(3):186–219, 2018.

[33] Haotong Cao, Han Hu, Zhicheng Qu, and Longxiang Yang. Heuristic solutions of virtual network embedding: A survey. *China Communications*, 15(3):186–219, 2018.

[34] Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, et al. BASMATI: Cloud Brokerage Across Borders for Mobile Users and Applications. In *Springer Advances in Service-Oriented and Cloud Computing Workshop*, 2018.

[35] Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Matteo Mordacchini, et al. Self-optimising Decentralised Service Placement in Heterogeneous Cloud Federation. In *IEEE International Conference on Self-adaptive and Self-organizing Systems (SASO)*, 2016.

[36] Xiang Cheng, Sen Su, Zhongbao Zhang, et al. Virtual Network Embedding through Topology-aware Node Ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[37] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[38] Ashish Cherukuri, Enrique Mallada, and Jorge Cortés. Asymptotic convergence of constrained primal–dual dynamics. *Systems & Control Letters*, 87:10 – 15, Jan. 2016.

[39] M. Chiang and T. Zhang. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, Dec 2016.

[40] Mung Chiang, Sangtae Ha, Fulvio Risso, Tao Zhang, and I Chih-Lin. Clarifying fog computing and networking: 10 questions and answers. *IEEE Communications Magazine*, 55(4):18–20, 2017.

[41] NM Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *IEEE INFOCOM 2009*, pages 783–791. IEEE, 2009.

[42] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. Almost optimal virtual machine placement for traffic intense data centers. In *Proc. of IEEE INFOCOM*, pages 355–359, 2013.

[43] G Dantzig and Delbert Ray Fulkerson. On the max flow min cut theorem of networks. *Linear inequalities and related systems*, 38:225–231, 2003.

[44] Francesco De Pellegrini, Francescomaria Faticanti, Mandar Datar, Eitan Altman, and Domenico Siracusa. Optimal blind and adaptive fog orchestration under local processor sharing. In *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, pages 1–8. IEEE, 2020.

[45] Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

[46] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[47] Tarek Elgamal, Atul Sandur, Phuong Nguyen, Klara Nahrstedt, and Gul Agha. Droplet: Distributed operator placement for iot applications spanning edge and cloud resources. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 1–8. IEEE, 2018.

[48] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic. In *2011 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2011.

[49] Francescomaria Faticanti, Francesco De Pellegrini, Domenico Siracusa, Daniele Santoro, and Silvio Cretti. Cutting throughput with the edge: App-aware placement in fog computing. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 196–203. IEEE, 2019.

[50] Francescomaria Faticanti, Francesco De Pellegrini, Domenico Siracusa, Daniele Santoro, and Silvio Cretti. Throughput-aware partitioning and placement of applications in fog computing. *IEEE Transactions on Network and Service Management*, 17(4):2436–2450, 2020.

[51] Francescomaria Faticanti, Lorenzo Maggi, Francesco De Pellegrini, Daniele Santoro, and Domenico Siracusa. Fog orchestration meets proactive caching. In *To appear in Proceedings of IFIP/IEEE International Symposium on Integrated Network Management 2021 (AnNEt 2021 Workshop)*. IEEE.

[52] Francescomaria Faticanti, Daniele Santoro, Silvio Cretti, and Domenico Siracusa. An Application of Kubernetes Cluster Federation in Fog Computing. In *24th Conference on Innovation in Clouds, Internet and Networks (ICIN 2021)*. IEEE, 2021.

[53] Francescomaria Faticanti, Marco Savi, Francesco De Pellegrini, Petar Kochovski, Vlado Stankovski, and Domenico Siracusa. Deployment of application microservices in multi-domain federated fog environments. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6. IEEE.

[54] Ana Juan Ferrer, Francisco Hernandez, Johan Tordsson, et al. OPTIMIS: A Holistic Approach to Cloud Service Provisioning. *Elsevier Future Generation Computer Systems*, 28(1):66–77, 2012.

[55] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. In *Classic papers in combinatorics*, pages 243–248. Springer, 2009.

[56] Y. Gan and C. Delimitrou. The Architectural Implications of Cloud Microservices. *IEEE Computer Architecture Letters*, 17(2):155–158, 2018.

[57] Yu Gan and Christina Delimitrou. The architectural implications of cloud microservices. *IEEE Computer Architecture Letters*, 17(2):155–158, 2018.

[58] X. Gao, X. Huang, Y. Tang, Z. Shao, and Y. Yang. Proactive cache placement with bandit learning in fog-assisted IoT systems. In *Proc. of IEEE ICC*, pages 1–6, 2020.

[59] Nam Ky Giang, Michael Blackstock, Rodger Lea, and Victor CM Leung. Developing iot applications in the fog: a distributed dataflow approach. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 155–162. IEEE, 2015.

[60] Yang Guo, Alexander L. Stolyar, and Anwar Walid. Online algorithms for joint application-vm-physical-machine auto-scaling in a cloud. *SIGMETRICS Perform. Eval. Rev.*, 42(1):589–590, June 2014.

[61] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques

in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[62] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2021.

[63] Bing Han, Jimmy Leblet, and Gwendal Simon. Hard multidimensional multiple choice knapsack problems, an empirical study. *Computers & operations research*, 37(1):172–181, 2010.

[64] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[65] Mhand Hifi, Mustapha Michrafy, and Abdelkader Sbihi. Heuristic algorithms for the multiple-choice multidimensional knapsack problem. *Journal of the Operational Research Society*, 55(12):1323–1332, 2004.

[66] I Hou, Tao Zhao, Shiqiang Wang, Kevin Chan, et al. Asymptotically optimal algorithm for online reconfiguration of edge-clouds. In *Proc. of ACM Mobihoc*, Paderborn, Germany, July 5–8 2016.

[67] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Cloud migration research: a systematic review. *IEEE Trans. on Cloud Computing*, 1(2):142–157, 2013.

[68] Joe Wenjie Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. Joint VM placement and routing for data center traffic engineering. In *Proc. of IEEE INFOCOM*, volume 12, pages 2876–2880, 2012.

[69] Khashayar Kamran, Edmund Yeh, and Qian Ma. Deco: Joint computation, caching and forwarding in data-centric computing networks. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 111–120, 2019.

[70] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American mathematical society*, 4(3):502–506, 1953.

[71] Petar Kochovski, Vlado Stankovski, Sandi Gec, Francescomaria Faticanti, Marco Savi, Domenico Siracusa, and Seungwoo Kum. Smart contracts for service-level agreements in edge-to-cloud computing. *Journal of Grid Computing*, pages 1–18, 2020.

[72] Bernhard Korte and Jens Vygen. *Approximation Algorithms*. Springer, 2012.

[73] Harold J. Kushner and Dean S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.

[74] Harold J. Kushner and G. George Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003.

[75] G. Li, J. Wu, J. Li, K. Wang, and T. Ye. Service popularity-based smart resources partitioning for fog computing-enabled industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, pages 1–1, 2018.

[76] Suoheng Li, Jie Xu, Mihaela van der Schaar, and Weiping Li. Popularity-driven content caching. In *Proc. of IEEE INFOCOM*, pages 1–9, 04 2016.

[77] X. Li, H. Ma, F. Zhou, and X. Gui. Service Operator-Aware Trust Scheme for Resource Matchmaking across Multiple Clouds. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1419–1429, 2015.

[78] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88, 2009.

[79] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proc. of IEEE SC*, Seattle, WA, USA, November 12–18 2011.

[80] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *2017 IEEE Fog World Congress (FWC)*, pages 1–6. IEEE, 2017.

[81] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[82] Florian Metzger, Tobias Hossfeld, André Bauer, Samuel Kounev, and Poul Heegaard. Modeling of aggregated IoT traffic and its application to an IoT cloud. *Proceedings of the IEEE*, 107:679 – 694, 03 2019.

[83] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497 – 1516, 2012.

[84] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, et al. Microservice architecture: aligning principles, practices, and culture. O'Reilly Media Inc., 2016.

[85] Arkadi Nemirovski. Advances in convex optimization: conic programming. In *Proc. of ICM*, Madrid, Spain, August 22–30 2006.

[86] Li-Na Ni, Jin-Quan Zhang, Chun-Gang Yan, and Chang-Jun Jiang. A heuristic algorithm for task scheduling based on mean load on grid. *Journal of computer science and technology*, 21(4):559–564, 2006.

[87] Y. Niu, F. Liu, and Z. Li. Load balancing across microservices. In *Proc. of IEEE INFOCOM*, Honolulu, Hawaii, USA, April 15–19 2018.

[88] Yipei Niu, Fangming Liu, and Zongpeng Li. Load balancing across microservices. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 198–206. IEEE, 2018.

[89] James M Ortega and Werner C Rheinboldt. *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.

[90] Roberto G Pachecom and Rodrigo S Couto. Inference time optimization using branchynet partitioning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2020.

[91] Claus Pahl and Brian Lee. Containers and clusters for edge cloud architectures–a technology review. In *2015 3rd international conference on future internet of things and cloud*, pages 379–386. IEEE, 2015.

[92] Boaz Patt-Shamir and Dror Rawitz. Vector bin packing with multiple-choice. *Discrete Applied Mathematics*, 160(10-11):1591–1600, 2012.

[93] Danny Raz, Itai Segall, and Maayan Goldstein. Multidimensional resource allocation in practice. In *Proceedings of the 10th ACM International Systems and Storage Conference*, pages 1–10, 2017.

[94] Benny Rochwerger, David Breitgand, Eliezer Levy, et al. The Reservoir Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development*, 53(4):1–4, 2009.

[95] Amit Samanta, Yong Li, and Flavio Esposito. Battle of microservices: Towards latency-optimal heuristic scheduling for edge computing. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 223–227. IEEE, 2019.

[96] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti. Foggy: A platform for workload orchestration in a fog computing environment. In *Proc. of IEEE CloudCom*, pages 231–234, Dec 2017.

[97] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[98] Marco Savi, Daniele Santoro, Katarzyna Di Meo, et al. A Blockchain-based Brokerage Platform for Fog Computing Resource Federation. In *Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2020.

[99] Robert-Steve Schmoll, Tobias Fischer, Hani Salah, and Frank HP Fitzek. Comparing and evaluating application-specific boot times of virtualized instances. In *2019 IEEE 2nd 5G World Forum (5GWF)*, pages 602–606. IEEE, 2019.

[100] Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and trends in Machine Learning*, 4(2):107–194, 2011.

[101] Mark Shifrin, Rami Atar, and Israel Cidon. Optimal scheduling in the hybrid-cloud. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 51–59. IEEE, 2013.

[102] R Sudeepa and HS Guruprasad. Resource allocation in cloud computing. *International Journal of Modern Communication Technologies & Research*, 2(4):19–21, 2014.

[103] Mohit Taneja and Alan Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *Proc. of IFIP/IEEE IM*, pages 1222–1228, 2017.

[104] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[105] Rahul Urgaonkar, Shiqiang Wang, Ting He, Murtaza Zafer, Kevin Chan, and Kin K. Leung. Dynamic service migration and workload scheduling in edge-clouds. *Perform. Eval.*, 91(C):205–228, September 2015.

[106] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[107] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[108] Eberhard Wolff. *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016.

[109] Andreas Wolke, Martin Bichler, and Thomas Setzer. Planning vs. dynamic control: Resource allocation in corporate clouds. *IEEE Trans. on Cloud Computing*, 4(3):322–335, 2014.

[110] Zhen Xia, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Trans. on Parallel and Distributed Systems*, 24(6):1107–1117, 2013.

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

[112] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM, 2015.

[113] Lei Ying, R. Srikant, and Xiaohan Kang. The power of slightly more than one sample in randomized load balancing. *Math. Oper. Res.*, 42(3):692–722, August 2017.

[114] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[115] R. Yu, V. T. Kilari, G. Xue, and D. Yang. Load balancing for interdependent iot microservices. In *Proc. of IEEE INFOCOM*, Paris, France, 29 April – 2 May 2019.

[116] Ruozhou Yu, Vishnu Teja Kilari, Guoliang Xue, and Dejun Yang. Load balancing for interdependent iot microservices. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 298–306. IEEE, 2019.

[117] Ruozhou Yu, Guoliang Xue, and Xiang Zhang. Application provisioning in Fog Computing-enabled Internet-of-Things: a network perspective. In *Proc. of INFOCOM*, 2018.

[118] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *International Journal of Communication Systems*, 26(8):1054–1073, 2013.