laszlo.toka_36_22

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Resource provisioning in cloud systems

DSc theses

Author
Laszlo Toka

January 26, 2023

laszlo.toka_36_22

Contents

1 Introduction **1**

1.1 Research topics around the cloud 1

1.2 Contributions and methodology 2

1.3 Structure of the Dissertation 2

2 Inter-cloud businesses **5**

2.1 Introduction 5

2.2 Business network model and formation game 7

2.2.1 Game definition 9

2.2.2 The Stackelberg game of Bertrand competitions 12

2.3 The pricing game in the cloud resource market 14

2.3.1 Cloud resource market model: actors, resources, strategies . . 14

2.3.2 Equilibrium prices in pricing games 18

2.3.3 Analysis of infinite-capacity games 25

2.4 Related work 28

2.4.1 Resource pricing in the cloud 28

2.4.2 Network formation models 30

2.4.3 Internet economics 31

3 Resource provisioning within cloud-based systems **33**

3.1 Introduction 33

3.2 Scalable and economical edge scheduling for latency-, and operation-critical applications 34

3.2.1 Complextiy analysis 38

3.2.2 Approximation bound 40

3.3 Re-scheduler: an offline orchestrator to minimize provisioned backup resources 45

3.4 Providing scalability with node clustering 48

3.5 Machine Learning-based auto-scaling 52
3.5.1 Analytical models of auto-scaling methods 54
3.5.2 The proposed proactive scaling engine 60
3.6 Operational model of running microservices 65
3.7 Related work 70
3.7.1 Orchestration of latency-critical cloud-native applications . . . 70
3.7.2 Auto-scaling solutions in the cloud 73

4 Bandwidth allocation in access networks towards the cloud 77
4.1 Introduction 77
4.2 An uncoordinated bandwidth sharing model and its analysis 78
4.3 A coordinated cloud resource optimizer method 83
4.3.1 System model and the resource allocation problem 84
4.3.2 Dynamic programming-based solution 86
4.4 Related work 89

5 Summary of scientific results 91
5.1 Inter-cloud business findings 91
5.2 Intra-cloud management and orchestration methods 93
5.3 Resource allocation of cloud access 95

Bibliography 97

List of Figures

2.1 Examples of the graph model of 5G infrastructure providers: the original service path between a selected pair of SAPs (*left*) and a new possible service path formed by establishing a new business connection between Tier-2 providers (*right*). 7

2.2 Graph model of operators' resources and service chains. 15

2.3 Resource allocation and service provisioning periods: In each time period (of a fixed length R) the served requests are those that arrived in the previous period. 18

2.4 Topologies with a single SAP, a 1 or 2 DCs and 0 or several NWs. . . 20

2.5 Equilibrium prices for various c, q values in the single NW topology. . 22

2.6 Equilibrium price $p^*(c)$ for $k \in [1, 10]$ providers and arrival q=0.1. Markers are used to describe the price for a finite capacity $c \in [1, 20]$ and a solid line for an infinite capacity $c = \infty$. 23

2.7 Equilibrium price $p_1^*(c_1, c_2)$ for provider capacities $c_1, c_2 \in [0, 20]$ and arrival $q = 0.1$. Markers are used to describe the price for finite capacities $c_i \in [0, 20]$ and a solid line for infinite capacities $c_i = \infty$. . . 24

2.8 Parallel paths with serial NWs. 25

2.9 The pine-tree topology. 26

2.10 Equilibrium price for NW nodes according to their distance from the DC in the top (x-axis: i, y-axis: p_i^*), and the same distance-dependent prices, but the sequential number of nodes are normalized by $k+1$ in the bottom (x-axis: $i/(k+1)$, y-axis: p_i^*) figure. One line corresponds to one numerical evaluation of formula in Lemma 2.13 for a given k value. 27

3.1 Proposed system architecture design for edge computing 35

3.2 The components of our edge-scheduler 36

3.3 Deterministic and non deterministic clustering examples 49

3.4 Measurement setup for application profiling 53

3.5 Profiles of different applications 54

3.6 One week Facebook traffic in a university campus (green: training data, red: test data, blue: weekend, not used) 58

3.7 One week of MMPP-based traffic generated from the Facebook load using the algorithm described in [99] (green: training data, red: test data, blue: weekend, not used) 58
3.8 Comparison of the lossy discrete model and HPA - Pod usage 59
3.9 Comparison of lossy discrete and lossless MMPP/M/c models - Pod usage 59
3.10 Simulations of discrete-time HPA and of the ML-based forecast models 62
3.11 Simulations of HPA's discrete-time and of the ML-based forecast models on the generated MMPP traffic 63
3.12 Proportion of HPA+ and HPA costs as a function of input load scale 65
3.13 Illustration of the scaling overhead model through an example application consisting of 5 modules 67
3.14 Modules of an illustrative example application, ordered by their scaling factor, and grouped (denoted by various colors and dashed line contours) 68

4.1 The Boosting Framework in a toy example for 2 users 78
4.2 Toy example for utility-based bidding policy (left-hand side) and zero bidding policy (right-hand side) 81
4.3 Evolution of number of players of various strategies 82
4.4 System overview 84
4.5 Example of utility functions and QoS, which are the expected number of detected events as a function of bandwidth. 85
4.6 The dynamic program for $B = 2$ 87

Chapter 1

Introduction

1.1 Research topics around the cloud

Cloud computing has transformed the scene of information technologies in less than two decades. The amazing technological evolution both in terms of computing and networking enabled several new applications and services running at extremely large scale on top of different cloud platforms. Public cloud platforms, such as Amazon Web Services [9], Google Cloud Platform [40] and Microsoft Azure [80] are capable of providing an "arbitrary" number of virtual resources on demand making use of virtualization techniques and resource management mechanisms. Well-designed data centers contain all the necessary physical assets, including thousands of blade servers and network devices, and the burden of operation is delegated to the cloud providers, ensuring high reliability and high performance. The cloud application owner has nothing else to do except for selecting the best-suited cloud service offering and deploying their application in the cloud to go live: this means i) zero initial investment as cloud services offer pay-as-you-go schemes, and ii) that there is no need to plan for maximum capacity as resource provisioning is flexible, usually automatically scaled, often assumed to be completely elastic. Besides the non-existent capital expenditures into infrastructure on the cloud tenant's side, deploying applications into the cloud also has the benefit of low operating costs. The reason behind the effective operation is the economies of scale of compute infrastructure in data centers, and the shared resources among a myriad of tenants, resulting in a time multiplexed usage of resources.

However, during the last decade, the centrally placed physical resources started to move closer to the users in order to enable the operation of novel types of applications, specifically latency sensitive ones. This paradigm shift has opened the door for telecommunications operators, mobile and fixed network vendors: they have joined the ecosystem to be part of the success story. Various concepts and paradigms appeared to designate the proper way how to leverage computing resources deployed in the vicinity of customers and end devices. Edge computing, fog computing, Multi-Access (formerly called as Mobile) Edge Computing, cloudlets are distinct concepts, nevertheless they share several common objectives and features [102, 76, 75, 142, 82, 83, 50]. Different technological and business use cases are

addressed by these concepts but the telecommunications stakeholders are crucial players in all scenarios as standalone entities or federated with cloud providers.

In addition, the emerging cloud platforms and the exposed capabilities have transformed the *software* running atop and also changed the corresponding software development techniques. Starting from monolithic applications running in dedicated Virtual Machine (VM)s, microservices have emerged: consisting of loosely coupled, inter-communicating software modules running in separate containers or as distinct functions managed by the underlying cloud system. At the end of the day, developers and service providers can obviously benefit from this shift, however, several new challenges arise on the platform side. Therefore, the research community has dedicated significant efforts to this topic during recent years and a large number of theoretical results have been published addressing different aspects and variants of the related mathematical problems: various techniques of several scientific fields were applied from mathematical programming across graph theory to machine learning.

1.2 Contributions and methodology

The goal of the cloud-related research presented in this Dissertation is to design and propose systems and methods that make the world of cloud computing even more reliable, affordable and easier to access for clients all over the world. The applied approach to reach this goal is two-fold: first, the involvement of traditional telecommunications operators in the cloud business is investigated, second, cloud management techniques are proposed for an enhanced Quality of Service (QoS). Within both approaches, end-to-end delay of cloud-based applications and an economical use of resources receive high emphasis: the former is indispensable for a good QoS, and the latter makes cloud applications even cheaper to operate.

The applied modeling tool set and analytical methodology covers a wide range of mathematical apparatus. Graph theory is used in modeling situations where the interplay of certain entities not only affects each other's status, but they do have indirect impact on those that are more than one hop away from them. Game theory is applied for use cases in which multiple rational entities interact, each pursuing their own agenda and predefined utilities, calling for a distributed framework to find stable points in the system. Machine Learning (ML) techniques ensure that the behavior of clients is accurately predicted, which is of paramount importance for reaching our aforementioned goals, i.e., providing reliable, affordable and accessible cloud services at the desired QoS level. Furthermore, the following chapters contain probability theory, queuing theory, differential equation systems, dynamic programming formulation, and Karp reductions to NP-complete problems.

1.3 Structure of the Dissertation

In Chapter 2 we start our investigation by characterizing conceivable business structures for multi-provider cloud-based service offerings. Assuming the current multi-tier Internet Service Provider (ISP) structure for worldwide connectivity services as

a starting point, we examine why and how the current ISP structure may change into new business connections, i.e., we study economic inter-cloud relations.

Beginning from the layered ISP order, we describe a network formation game in which players represent the ISPs; link creation represent business associations with an upkeep cost; operational expenses and incomes are determined to be dependent on paid and received middleman costs offering mediator services. Our work is inspired by the body of related work addressing the phenomenon of growing number of peering relations between ISPs. In contrast to those work, our study focuses on agreements between 5G and cloud infrastructure providers, either directly or via mediating parties, instead of exchanging Internet traffic through either peering or transit relations studied in the related research papers.

Moreover, we believe that the current Internet pricing models based on transit and peering cost decisions will be challenged due to the change of the services obtained in the future. More specifically, as long as the service is mere connectivity, aggregation and hiding the connectivity topology are substantial. Therefore, a transit provider providing default access to the rest of the Internet is sufficient unless serious bottlenecks are detected. However, in envisioned 5G services connectivity interleaves with compute services where proximity becomes a key asset. Therefore, address reachability information may be complemented with cloud availability and capability information on considering edge deployments.

While 5G has been driving a revolution in the field of networking [41, 94], the ongoing and envisioned changes affect many other areas from cloud/edge computing to vertical industries including health care, Industry 4.0, and transportation [131]. In the visions of 5G the often heard service-level keywords are cost-effectiveness and improved service provisioning with fast creation, fast reconfiguration and with large geographical reach of customers. This paradigm shift is technologically enabled by Network Function Virtualization (NFV) [31], i.e., implementing telco functions in VMs that can be run on general purpose computers instead of running them on expensive dedicated hardware as the traditional manner, and by Software Defined Networking (SDN) [91], i.e., configuring and controlling network appliances with easily manageable, often centrally run software applications [78]. We study the economic aspects of this paradigm shift: how providers set their prices in such a market, and how customers select the necessary resources given those prices.

Next, in Chapter 3, we pinpoint the most pressing challenges in cloud resource management, typically stressed in edge cloud settings, and we offer reassuring solutions to those.

Future applications, e.g., extended reality applications or 5G and beyond telco services, will require ultra-reliable and low-latency communication from the hosting compute and network infrastructure. Edge computing, built on the fast access network of 5G, is capable of fulfilling such strict delay criteria. But remote edge nodes are prone to failures and their downtime might be longer than that of a central infrastructure, i.e., data centers. Therefore, while the edge deployment of compute elements of the service minimizes service delay, ensuring the high reliability of services is a challenge.

Allocating resources dynamically to constituent VMs, containers and scaling them properly on demand can be a challenging task. The scaling logic can be driven by various service management goals, e.g., either minimizing resource usage while sustaining a given QoS target, or minimizing Service Level Agreement (SLA) violations no matter the price paid for the provisioned resources. The decision-making in cloud scaling is further aggravated by the peculiar scaling behavior of the managed application, i.e., the function that translates the amount of requests to be served respecting the given SLA constraints to the necessary amount of resources to provision.

Cloud-native applications typically follow the microservice architecture nowadays, where the monolithic software is broken down into smaller independently managed components, usually realized by software containers that are separately orchestrated and scaled by the cloud platform enabling optimal resource utilization [4]. This brings the possibility of extra delays in the application's response times, which must be tackled in order to sustain a high QoS.

These intra-cloud challenges are addressed in respective sections of Chapter 3. In the subsequent Chapter 4, we take a further step down, and study resource provisioning under the cloud: we propose service quality assurance frameworks in an uncoordinated and in the centrally optimized setting. In the first case the users get an opportunity to signal their urgent bandwidth demands and the scheduling decisions are made at the network access point based on those. In the second case we present a cloud-based IoT architecture that maximizes utility when high-bandwidth video streams have to share a narrow uplink channel. The scheme attains a significant reduction in the used uplink bandwidth and processing power in the cloud.

The organization of the Chapters 2, 3 and 4 follow the same structure: in each Chapter we provide a general introduction on the main problems first; next, we present formal models and problem formulations arising in the context of the tackled challenges; then we describe the main contributions; and finally we review related research and position the new results against the state-of-the-art. Each chapter stands on its own and can be read independently from the rest; notations and definitions are provided in-place.

Finally, Chapter 5 is devoted to a compact description of the most important findings in a formal representation.

Chapter 2

Inter-cloud businesses

2.1 Introduction

Keeping the activity of the autonomous systems manageable in the present Internet has a cost of lessening the degree of their interoperability to best effort. Subsequently, when one considers making services that reach over numerous administrative domains, they cannot guarantee Quality of Service (QoS). The 5G vision anticipates online services to be more advanced [17] than the present ones for which infrastructure providers ensure end-to-end QoS in *network slices* [30] with small delay and large bandwidth capacity. Moreover, with the appearance of virtualization both in compute and network technologies, quicker service creation is feasible and the reconfiguration of those can be more versatile, bringing about a totally novel life-cycle management approach contrasted with what the present standard is [110]. The idea of *elastic resource slicing* [129] is the key empowering ingredient for this [42], and when different providers partake in making a resource slice, comparably exacting commitment to QoS assurance is required from all members [129, 111].

The locality of clients is one of the most significant driving variables of the birth of *multi-provider resource slices* [129, 111]. Connectivity requisites of services naturally characterize the potential set of qualified networks for the resource slice: the required latency guarantees of a given Virtual Network Function (VNF) decide whether it should be deployed to the cloud or to an edge cloud (or fog) near the clients, and in which one they can be sent for conveying QoS [90]. One of the other significant characteristics is cost: when QoS necessities permit, it can drive the requirement for multi-domain resource slice creation [125]. Luckily, Network Function Virtualization (NFV) makes it conceivable to make adaptable services in the form of a Service Function Chain (SFC) of VNFs when the suitable resource slice has been made.

Roused by these elements, we anticipate that 5G framework actors will work together in alliances. In order to have the option to offer locality-aware services for their clients around the world, they will utilize the compute and network assets of any of their fellow suppliers [129, 111]. However, before committed resource slices are provisioned crossing over different providers' authoritative domains, business arrangements must be set up between partners, e.g., on cost and QoS assurance designated to the resource slice [49, 120]. For the end-to-end QoS capabilities of

a resource slice that numerous providers partake in, dispersed arrangements and consistently maintained business connections may be vital among the stakeholders [27].

If incumbent transit providers, who do not necessarily possess (deep) edge clouds, are not willing to proxy, process, and forward network slice orchestration requests, then providers must establish bi-, or multilateral contracts similar to Internet peering, which come with extra operational costs. We investigate the rationale for all operator players to evolve services from connectivity-only to 5G network slicing via their existing transit and peering business relationships, i.e., under what mediation pricing strategy incumbent transit providers can enter the 5G slicing market to prevent lower tier providers to enter into bilateral direct business contracts. We research how the organization of business relations may advance in the reign of 5G: will it follow the geography of transit and peering relations of the Web today [89] or will new business relations be set up between neighboring or far off providers correspondingly to ISPs' peering arrangements?

Due to this novel flexibility of the control and management of networking, new types of actors will enter the telco market, resulting in a largely heterogeneous ecosystem with currently unexplored business relationships among the stakeholders. For example, traditional telcos with physical resources and geographical footprint can provision basic Infrastructure as a Service (IaaS) or Network Function Virtualization Infrastructure as a Service (NFVIaaS) in the new context for remote telcos without footprint, or for Over-the-Top (OTT) solution providers. But they can also extend their product range with Software as a Service (SaaS) offerings in order to step up in the value chain and to increase potential revenue. Alternatively, software solutions can be delivered by third party VNF developers, providing VNFaaS in this context even for the telco itself. We argue therefore that the new NFV ecosystem fundamentally redefines how telecommunications enterprises will soon operate not only from a technical, but also from a business perspective.

Online services can be best implemented as SFCs [46] in which functions are run separately, possibly in remote data centers, while network control ensures connectivity between those and the end users. This tightly integrated environment encompassing cloud and network resources, together with the extreme service level requirements of verticals' use cases, poses serious technological challenges on all stakeholders. The related issues seem hot topics and are being addressed by several research projects and working groups of standardization bodies [141, 63]. However, in our opinion the economic aspects and business related challenges have not received enough attention yet. We therefore establish a novel economic framework in order to understand the NFV ecosystem, to evaluate analytical models, and to propose novel pricing schemes and co-operation strategies among stakeholders. We tackle the analysis of the expected business interaction between the infrastructure providers and the customers. More specifically, we are interested in how providers set their prices in such a market, and how customers select the necessary resources given those prices. We argue that price is an essential, if not the most important attribute of resources, therefore it is crucial to take it into account in resource orchestration methods.

2.2 Business network model and formation game

We formalize the development of business relations as a network formation game in an envisioned multi-provider multi-user setup, where resource slices are created for users that might be located out of the direct reach of their provider, thus the slices to be created are possible overarching multiple providers, comprising compute and network resources of remote providers. We present the interplay between two contradicting effects. First, we represent a creation and upkeep cost of business connections that are paid as a provider builds up contracts with a number of fellow actors. Second, we introduce the notion of mediation prices, collected by middleman actors; these providers who have direct links to other providers that are not directly interconnected with a business link, but want to initiate the creation of a resource slice with each other, with the help of the middlemen.

We study this game from the perspective of profit-oriented 5G infrastructure providers in order to characterize this business trade-off of those opposing forces. Our contributions consist of (i) analytic conditions on these pricing factors for a stable business relationships topology, (ii) formulas that describe the expected middleman prices if providers start making business relationships from scratch. For the first point, we write a formal condition on the highest price that middlemen can ask, at which the actors do not have any incentives to create new business links.

As analytical results, we prove that high tier providers have a motivating force to keep their broker costs low in order to preserve their place in the ecosystem, and in an alternative setup with business links being created starting from a clean slate situation, we determine the equilibrium prices with parameterized formulas. With the numerical results, we show the interesting interplay of the number of created links and the attached costs, in addition to the distribution of business path lengths that arise in stable setups.

The rest of the section is organized as follows. We first define our business network model and the trade-off we study in detail; then, we formalize the model as a network formation game, and derive analytic results on equilibrium conditions.

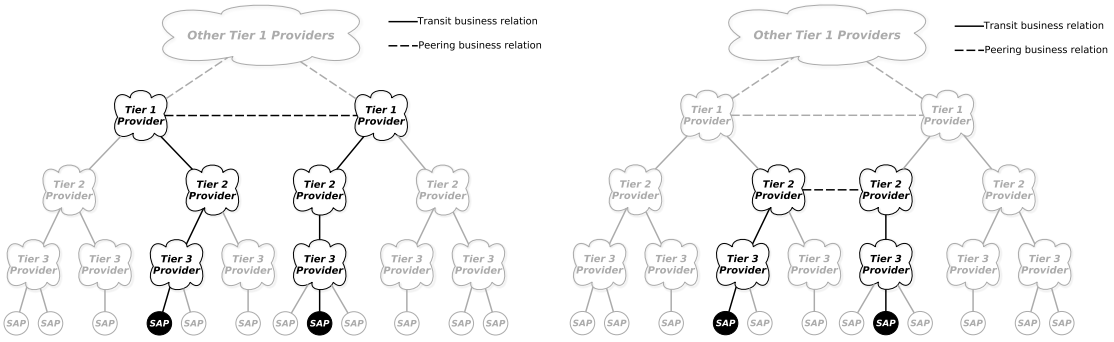


Figure 2.1: Examples of the graph model of 5G infrastructure providers: the original service path between a selected pair of SAPs (*left*) and a new possible service path formed by establishing a new business connection between Tier-2 providers (*right*).

Figure 2.1 shows instances of our graph model of 5G providers. In this model, vertices stand for (i) network providers (likewise offering computing services) and (ii) Service Access Points (SAP), which are reaching points of end-clients. Edges represent business connections, which, in the underlying stage can be either (i) transit relations (depicted by solid lines in the figure) or (ii) peering relations (depicted by dashed lines) between nodes. The locality of clients and delay-critical services are among the most significant driving components of multi-provider setups; accordingly, we portrayed SAPs only at Level 3 providers as those offer access service to end-clients [56]. However, in our vision all network providers are expected to become 5G cloud operators as well [129, 111].

In the center of our model are such multi-domain services in which the end-clients of the service are clients of provider A, but they really utilize the service inside the access domain of provider B. In this situation, provider A purchases a resource slice from provider B (and perhaps from different providers interconnecting providers A and B). We model these business arrangements as paths in the graph model associating two nodes: the purchaser of the service is essentially the customer-facing provider, while the seller of the service is the provider in whose domain end-clients currently utilize the service. The delay-critical service is subsequently placed at the vendor provider's premises near the current SAP of the end-client. The service's descriptive characteristics, e.g., computing needs, explicitly stating VNFs to onboard that comprise the service, network QoS to the SAP, and so forth, should be satisfied and paid for by the purchaser provider. In this multi-provider arrangement, we investigate the development of business relations: we do not handle the pricing of services or that of the resource slices, this study is limited to the valuation of inter-provider organizational possibilities.

Deploying a service into the resource slice that is mounted on another provider(s) infrastructure requires prior negotiations and a pre-built business link or path among the partners, similarly to the network formation models, e.g., in [13]. We assume that two opposing effects decide how these arrangements are made: either by a direct business made between the two parties, or through a chain of agents that relay between the seller and purchaser providers. In the second case, the base cost of the resource slice is supplemented by the business relay cost of the interconnecting providers. The length of the chain of mediating providers equals to the number of hops on the path between the two providers in our graph model. Alternatively, each extra immediate business connection between providers incurs an expense at the two players: setting up and keeping up business contracts have their expenses. The trade-off situation is clear: while edges increment the general managerial expenses, barring mediators from setting up resource slices in a remote operator's infrastructure saves cost. Compared to the classical network formation games [34, 25, 116], the utility of being strongly connected to other nodes there, is, in our case, translated to fewer middlemen to pay for mediating the business between remote nodes.

Figure 2.1 showcases the business connection between a chosen pair of SAPs in an underlying layered graph model. The path traverses the transit links towards the highest level of the tiered structure of providers. Nevertheless, another business interconnection can be shaped by building up business associations between any

two nodes at lower tiers, e.g., between Tier 2 providers in this case, as appeared in Figure 2.1. We underline the important notice that the new business connection does not change the data plane path if we assume that the newly connected providers are not neighboring ASes. Tier 1 providers will presumably provide the data plane connectivity, which might be provisioned for the relating Tier 2 providers on an alternate timescale. The new business path is, however, abbreviated as Tier 1 providers do not act as business mediators any longer.

In the following, we characterize the network formation game we use for tackling the aforementioned model. As part of the study, we infer significant characteristics that portray the equilibria of the game.

2.2.1 Game definition

For tractability, let us use the notation of [25]. We consider the network service providers as the players of a network formation game. N denotes the player set $\{1, 2, \ldots, n\}$. S_i denotes the strategy set of player i, which is the power set of $N \setminus i$, in other words, the collection of possible sets of other players to create a link with. s_{ij} indicates the whether i wants a link between nodes i and j, so $S_i = \{(s_{ij})_{j \neq i} | s_{ij} \in \{0, 1\}\}$ and $|S_i| = 2^{n-1}$. The strategy of i is s_i and $s_i \in S_i$. The combination of the strategies of all players provide the outcome of the game. The resulted strategy profile is denoted by $s = (s_1, s_2, \ldots, s_n) \in S_1 \times S_2 \times \cdots \times S_n$. The outcome of this one-shot game is an undirected graph $G(s) = (N, E(s))$ in which a given edge is built if there is consent between the two nodes, i.e., $E(s) = \{(i, j) : i \neq j, s_{ij} = 1 \wedge s_{ji} = 1\}$. That is, both players i and j must agree to establish a link between each other in order for it to be created.

Cost function c determines the player cost given the strategy profile out of the combination of strategy sets, i.e., $c : S_1 \times S_2 \times \cdots \times S_n \to \mathbb{R}^n$. As in related work, the expense brought onto player i when all players embrace strategy s is comprised by the expense based on the quantity of links $|s_i|$ that player i sets up effectively with different nodes, and by the aggregate of the agent expenses paid to middlemen to reach every single other node. As a novel term, we represent the salary that is created by mediating business going through node i. In our game, the total cost is defined as follows:

$$c_i(s) = \alpha |s_i| + \sum_{j \in N \setminus i} \beta d_{(i,j)}(G(s)) M_{i,j} - \sum_{j \in N \setminus i} \sum_{k \in N \setminus i,j} \beta \mathcal{I}_{i \in p_{j,k}}(G(s)) M_{j,k} \quad \forall i \in N, \tag{2.1}$$

where α and β are the business peering cost and the middleman price, respectively; $d_{(i,j)}(G(s))$ denotes the number of middlemen on the shortest-path between providers i and j in the business graph G; $M_{i,j}$ depicts the extent of services bought by i from j through whatever path of middlemen providers this business is realized; and $\mathcal{I}_{i \in p_{j,k}}(G(s))$ indicates whether i is on path $p_{j,k}$, i.e., the shortest path between j and k. If no path exists between i and j, then $d_{(i,j)}(G(s)) = \infty$. Therefore, the first term of the cost formula stands for the business link creation, the second term reflects the price to pay middlemen for reaching providers indirectly, and the third term is the income that is generated by acting as middlemen for other providers' businesses.

As in [25], this game model reflects a setting in which building connections is expensive; however, an extensive direct network might be beneficial to build in order to limit the brokers to pay off. Likewise, the more connections a provider has, the more probable it will earn as a middleman, subsequently lowering the total cost. Naturally, providers try to limit their costs characterized in Equation (2.1). If the expense of an extra business link α, the mediator cost β, and the business demand M are all fixed, the game comes down to the question of which new connections are worth being made so as to reduce expenses.

For different sorts of equilibrium, stability conditions, lower and upper limits on the price of anarchy in network formation games, we refer the reader to [34, 25, 116]. Note that our model is different from the models found in the related work. The game variant that is the most similar to ours was introduced in [13]; however, contrary to that model where a player's transit traffic brings about expense, in our arrangement, the more shortest paths that traverse a node, the more income is created to that respective player. For their arrangement, the authors of [13] demonstrated that the steady result of the game is consistently a tree, as more transit paths and the edge creation are not worth making lower distances to different nodes, once the graph is connected. In our game, the final graph $G(s)$ can also be a tree for high connection creation cost α: both lower distance to different nodes, and mediating more business paths decrease the expense, hence, if edge creation is generally low cost, it is useful to make a few.

We expect an underlying layered topology of providers. The objective of the work introduced in this section is to give an adequate condition under which there are no new connections made by the players. In case this condition is fulfilled, the underlying layered structure is along the stable state of our game, i.e., an equilibrium.

Assumption 2.1. There are business links between providers originally, and these links organize nodes in a tiered topology, denoted by G^0, such as the one depicted in Figure 2.1.

Let us number the tiers from top to bottom, $1, 2, \ldots, t$, and let t^i indicate the tier that provider i belongs to. Let C_i denote the set of providers that can be reached downwards in the tiered topology through provider i, i.e., $C_i = \{k \mid i \in p_{j,k}\ \forall j \mid t^j = 1, t^k > t^i\}$, preferring peering links in Tier-1 to peering links in lower tiers. Now, we deduce the parametric cost saving when a new link is created.

Lemma 2.1. *If Assumption 2.1 holds, the highest cost reduction a new link between two nodes, i and j, can result in is*

$$2\beta(t^i + t^j - 2)|C_i||C_j| \max\left(M_{kl|k\in C_i, l\in C_j}, M_{kl|l\in C_i, k\in C_j}\right) - 2\alpha \tag{2.2}$$

Proof. Providers i and j, belonging to tiers t^i and t^j respectively, would both make a cost reduction for their children in C_i and C_j by interconnecting themselves with a new link and thus lowering the second term of Equation (2.1) of the children. At most, $t^i - 1 + t^j - 1$ middlemen in upper tiers are shortcut from cross paths between the two sets of children with the new link. This number might be lower if any peering links exist between parents of i and j, or if they have the same Tier-1 parent. Note that we assume full mesh among Tier-1 providers in G^0. The cost

allocated to middlemen is proportional with the extent of the business which is upper bounded by $\max(M_{kl|k\in C_i,l\in C_j}, M_{kl|l\in C_i,k\in C_j})$. The number of business relationships is given by $|C_i||C_j|$, hence the result starting from the following formula:

$$\sum_{k\in C_i}\sum_{l\in C_j}\beta(t^i-1+t^j-1)(M_{kl}+M_{lk})-2\alpha.$$

□

Hindered by the complexity in a general tiered topology setting, we make the following assumption on the number of children each node has, and of businesses leaf nodes make.

Assumption 2.2. G^0 contains a number of Tier-1 nodes connected in full mesh, and a tree subgraph under each Tier-1 node in which intermediary nodes have at least k children, and all leaf nodes are at the same depth t. Furthermore, any pair of leaf nodes exchange m amount of business; intermediary nodes do not act as service sellers or buyers.

Given the specific tiered topology of Assumption 2.2, we prove that the highest cost saving can be attained with new peering links in the topmost tier.

Lemma 2.2. *Under Assumption 2.2, the higher tier the nodes belong to, the larger the cost saving that is attained if they create a new link.*

Proof. Under Assumption 2.2, the size of C_i and C_j are lower bounded by the number of leaves of perfect k-ary trees: $|C_i| \geq k^{t-t_i}$. The cost saving of two nodes i and j by creating a link is $2\beta(t^i+t^j-2)k^{2t-t^i-t^j}m-\ 2\alpha$, where m represents the amount of business any pair of leaf nodes exchange under Assumption 2.2. By expressing $x = t^i + t^j$, it is easy to see that this cost saving is higher when $\frac{x-2}{k^x}$ is larger. As $k \geq 2$ and $x \geq 3$, the maximum is attained if $x = 3$, i.e., $t^i = 1$ and $t^j = 2$, or $x = 4$, i.e., $t^i = 2$ and $t^j = 2$. □

We suppose that providers may turn to dynamic pricing schemes for middleman fees in order to bar the monetary incentives of new business link creation: as mentioned above, we seek the stable level of middleman prices when the status quo is kept, i.e., top tier providers demotivating low tier providers of making new business peerings. We determine this exclusionary price level in the assumed topologies.

Theorem 2.1. *Under Assumption 2.2, if all providers keep their price below* $\frac{\alpha}{k^{2t-3}m}$, *then topology* G^0 *is an equilibrium.*

Proof. It is easy to see that the topmost tiers lose business if peerings are created underneath them. In such a topology G^0 that satisfies Assumption 2.2, the maximal middleman price β for which no new links are worth being created between any two providers is given by $2\beta(t^i+t^j-2)k^{2t-t^i-t^j}m-2\alpha \leq 0$ from which the upper bound on β is $\alpha\frac{k^x}{(x-2)k^{2t}m}$ with $x = t^i + t^j$, according to Lemma 2.2. A statement is given by $x = 3$. □

As a result of Theorem 2.1, the high tier providers have a motivating force to keep their broker costs low. The way that they need to save the status quo in terms of business relations among providers has a general constructive outcome on the whole system: the middleman cost of building up multi-provider businesses is upper limited. This bound is directed by the topology and the link creation cost.

2.2.2 The Stackelberg game of Bertrand competitions

Finally, in this section, we describe the equilibrium point of the network formation game with the assumption of initially non-existent peering relations between low tier providers, and of missing transit links between tiers.

Assumption 2.3. Let us assume a 3-tier topology of providers, initially with no transit/peering links other than the Tier-1 full mesh. Furthermore, we assume that one transit link can be built by each Tier-2 (to a Tier-1) and Tier-3 (to a Tier-2) provider for no cost.

Bertrand competitions [18] describe interactions among suppliers that set prices and their customers that choose quantities to purchase from them at the prices set. The Bertrand competition model assumes that (i) there are at least two suppliers producing a homogeneous (undifferentiated) product and cannot cooperate in any way, (ii) suppliers compete by setting prices simultaneously and consumers want to buy everything from a supplier with a lower price (since the product is homogeneous and there are no consumer search costs), (iii) if suppliers charge the same price, consumers' demand is split evenly between them, (iv) all suppliers have the same constant unit cost of the product or service, so that marginal and average costs are the same, and (v) each supplier has sufficient capacity to serve all customers.

Bertrand proved that, if suppliers chose prices strategically, then the competitive outcome would occur with the equilibrium price equal to marginal cost. In our specific case, the product to sell is the middleman service; therefore, we suppose that there is no maximum capacity imposed on the suppliers, and they all offer the same service.

We argue therefore that the game of setting individual middleman prices in Tier-1 and Tier-2 are Bertrand competitions , as lower tier providers seek to build their transit link to one with the lowest price. Consequently, the Nash Equilibrium of the Bertrand game in Tier-1 results in a homogeneous β_1^* middleman price for all Tier-1 providers, covering the marginal cost of the full mesh linkage. Considering the average transit business they take care of, the following statement displays the value of β_1^*.

Lemma 2.3. *In equilibrium, Tier-1 providers all set their middleman prices to* $\beta_1^* = \frac{n_1(n_1-1)\alpha}{2\gamma m}$*, where* n_1 *is the number of Tier-1 providers,* γ *is the fraction of businesses reaching Tier-1 and* m *is the grand sum of business matrix* M*.*

Proof. As in a Bertrand duopoly competition, the only equilibrium price for the n_1 competing providers is at the marginal cost, since any provider setting a higher price would lose its customers. The total income is provided by the fraction of

businesses flowing through Tier-1 providers, i.e., not via Tier-2 peering links, denoted by $\gamma m 2\beta_1$. Supposing a uniform distribution of businesses flowing through the Tier-1 providers, and an equal share of peering costs, then applying β_1^* covers the cost of full mesh peering at each Tier-1 provider. □

As Lemma 2.3 shows, the Bertrand game equilibrium is partly defined by the fraction of business (γm) that will reach Tier-1 providers in the first place. This fraction is, in turn, dependent on the middleman price that Tier-1 providers set, i.e., β_1^* because, for certain Tier-2 providers that exchange relatively large amount of business, creating a direct link might be beneficial compared to paying the Tier-1 middlemen. In order to grasp this condition, we introduce the empirical distribution of such business amounts.

Definition 2.1 (Distribution of businesses)**.** Let us denote the empirical distribution of the amount of business between Tier-2 provider pairs by $f(\mu)$. Similarly, let us denote the distribution of those between Tier-3 providers by $g(\mu)$.

Both Tier-2 and Tier-3 providers have the option of creating peering links in case it is less costly than dealing with middlemen. As middleman prices grow, more and more provider pairs decide so. Therefore, the fraction of business from Tier-2 to Tier-1, and from Tier-3 to Tier-2, decreases with the rise of middleman prices: demand is monotone decreasing in β_1 and β_2, respectively. Furthermore, peering links are created in the decreasing order of the amount of business between the two endpoints, as the middleman price grows. This phenomenon creates a Stackelberg game [26] nature of the middleman price setup between Tier-1 (being the leaders) and Tier-2 (being the followers) . The Stackelberg competition suits well the situation as the leaders, i.e., Tier-1 providers move first by setting their middleman prices homogeneously, and then the followers, i.e., Tier-2 providers move sequentially, deciding about the link creation in function of Tier-1 prices. Let us see how we can deduce the equilibrium price in Tier-2.

Lemma 2.4. *In equilibrium, the Tier-2 providers' middleman price is*

$$\beta_2^* = \frac{\alpha}{2\delta m}\left(n_1(n_1-1) + n_2(n_2-1)\int_{\frac{2\gamma m}{n_1(n_1-1)}}^{\infty} f(\mu)d\mu\right),$$

where δ is the fraction of business reaching Tier-2, and n_2 denotes the number of Tier-2 providers.

Proof. Similarly to Lemma 2.3, in equilibrium, all Tier-2 providers apply the same middleman price β_2^*, and their total income precisely covers their cost. The overall Tier-2 income is given by the Tier-3 providers, with their business not traversing through their own peerings, i.e., $\delta m 2\beta_2$. The middleman fee and the creation of peerings constitute the total cost of Tier-2 providers, i.e., $\gamma m 2\beta_1^*$ and $2\alpha\int_{\mu_2^p}^{\infty} f(\mu)d\mu\frac{n_2(n_2-1)}{2}$, respectively, where μ_2^p denotes the amount of business over which the peering is cheaper than via Tier-1 middlemen. This latter condition gives $\mu_2^p = \frac{\alpha}{\beta_1^*}$. Substituting the value of β_1^* with $\frac{n_1(n_1-1)\alpha}{2\gamma m}$ from Lemma 2.3 yields the formula for β_2^*. □

Finally, we can draw the amount of business that Tier-3 providers will carry through Tier-2 providers. The following statement expresses the necessary formulas for substituting δ and γ in Lemmas 2.3 and 2.4.

Lemma 2.5. *In equilibrium, the fraction of business flowing through Tier-2 and Tier-1, respectively, are:* $\delta m = \int_0^{\frac{\alpha}{\beta_2^*}} \mu g(\mu) d\mu$, *and* $\gamma m = \int_0^{\frac{\alpha}{\beta_1^*}} \mu f(\mu) d\mu$.

Proof. Similarly to the proof of Lemma 2.4, let us denote by μ_3^p the amount of business over which the peering is cheaper than through Tier-2 middlemen. Analogously to μ_2^p, $\mu_3^p = \frac{\alpha}{\beta_2^*}$. The statement then follows. □

The equilibrium prices can be determined numerically if $n_1, n_2, f(\mu), g(\mu), m, \alpha$ are given. Moreover, if Tier-3 providers select their Tier-2 transit partner randomly, and the ratio between the number of Tier-2 and Tier-3 providers is sufficiently small, the central limit theorem might be applicable in order to determine $f(\mu)$ based on $g(\mu)$.

2.3 The pricing game in the cloud resource market

We now turn to the pricing of the services offered to end users. First, we introduce a model to describe the requirements of a service that a customer wants to deploy, and the attributes of the resources from which the customer selects. Building on the model, we define the provider-customer interaction as a game, and we formalize the resource selection problem. Second, we show analytically proven optimal pricing strategies for stochastic games in which the customer behavior is modeled as random process, and resources are finite or infinite. Third, we evaluate realistic network topologies and show relevant results with the assumption of abundant provider capacities.

2.3.1 Cloud resource market model: actors, resources, strategies

Our work tackles the trades of resources that are necessary for deploying SFCs: computing power in data centers, and network bandwidth from the end-users, modeled as service access points, to data centers.

We consider two types of actors in our model: the infrastructure (data center and network) providers and their customers, the latter possibly supplying application to the home user or to enterprises. The resources that are traded between these two types of actors are compute and network resources. Both types are offered with given capacity and tolling a given latency. Furthermore, we consider service access points as resources where the customers want to make their to-be-deployed service accessible to their customers. We build a graph model for these resources; an example illustration is shown in Figure 2.2.

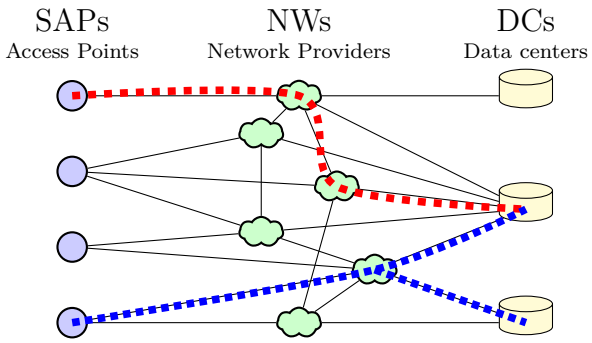


Figure 2.2: Graph model of operators' resources and service chains.

Let $\mathcal{A}$ denote the set of SAPs, $\mathcal{N}$ the set of network providers and $\mathcal{D}$ the set of data centers.

Definition 2.2 (Resources)**.** We define resources as an undirected graph $G = (V, E)$ where V denotes the set of nodes and E the set of edges. There is a node assigned to each SAP, each network provider and each data center in G, thus the total number of nodes is $|V| = |\mathcal{A}| + |\mathcal{N}| + |\mathcal{D}|$. The following forms of edges can appear (see Figure 2.2):

- SAP - network provider: each customer wants to create an online service that is reachable at one or more network providers;
- network provider - network provider: interconnect transit (Tier-1) and access (Tier-2 and Tier-3) networks that have direct links (representing peering) or connectivity at Internet Exchange Points, etc.;
- network provider - data center: each data center is connected to the Internet via at least one network provider.

As illustrated in Figure 2.2, the graph G has no edges between the nodes in $\mathcal{A}$ and no edges between the nodes of $\mathcal{D}$. Likewise, there are no edges connecting a node in $\mathcal{A}$ with a node in $\mathcal{D}$.

We denote the providers' capacity with c, which can be either a network capacity c_n for $n \in \mathcal{N}$, or compute capacity in a data center c_d for $d \in \mathcal{D}$. We introduce l_n for $n \in \mathcal{N}$ and l_d for $d \in \mathcal{D}$ as the latency (i.e., delay) a network provider n and a data center provider d guarantees, respectively. For pricing, we use the notation p for the current price of resource units for both network and data center providers. Being part of the customer-facing network operators, the price is zero for SAP nodes, i.e. $p_a = 0$, $\forall a \in \mathcal{A}$, and those abstract nodes do not raise any technical obstacles, i.e., $c_a = \infty, l_a = 0$, $\forall a \in \mathcal{A}$. The details are summarized in Table 2.1.

Customers arrive with random demand for resources which are used for deploying a service for the customer's end users. Demand characteristics include specific service access points, certain amount of computation and network capacity, maximum end-to-end latency and maximum total cost. Formally:

	Components			Service requests	
	SAPs	NWs	DCs	instance	random variable
	$\mathcal{A}$	$\mathcal{N}$	$\mathcal{D}$	s	S
Capacity	∞	c_n	c_d	c_s	C
Latency	0	l_n	l_d	l_s	L
Price/budget	0	p_n	p_d	b_s	B

Table 2.1: Summary of the applied parameters.

Definition 2.3 (Service request)**.** A service request is defined by a 4-tuple $s = (a_s, c_s, l_s, b_s)$, describing

1. a selected access point $a_s \in \mathcal{A}$,
2. a required resource amount c_s (capacity),
3. an upper limit l_s on the end-to-end network latency that has to be met from SAP to DC,
4. a budget b_s that is an upper limit on the total cost the customer is willing to pay.

The request refers to a single time period as defined later. For the sake of simplicity, we assume in our model, that the amount of resources required by a request is characterized by a single parameter describing for instance either a measure of computation required to run the service in DCs, or corresponds to consumed network bandwidth at NWs. We do not consider multidimensional combinations of this parameter. This implies that any DC can serve any service, i.e., the computation never requires any special hardware. Notation is summarized in Table 2.1.

Here we define the business interactions in the market, i.e., how deals are made.

Definition 2.4 (Resource allocation)**.** A service request s is served through a path t or a set of paths $\mathcal{T}$ in G.

We first describe a service through a single path. A path t is a sequence of nodes $(a_t, n_{t,1}, \ldots, n_{t,\ell(t)}, d_t)$ where $a_t \in \mathcal{A}, d_t \in \mathcal{D}$ and the number of its network providers is denoted by $\ell(t)$. The latency of the path is defined as the sum of the latencies of its network providers and the latency of its data center $l_t = \sum_{i=1}^{\ell(t)} l_{n_{t,i}} + l_{d_t}$. The path is associated with the unit price $p_t = \sum_{i=1}^{\ell(t)} p_{n_{t,i}} + p_{d_t}$ and an allocated capacity c_t. The allocated capacity must take into account the capacity of the resources along the path and the capacity allocated to other paths using them. To serve a request s the path t has to satisfy $a_t = a_s$, $c_t = c_s$, $l_t \leq l_s$, $c_t p_t \leq b_s$.

A request can also be served by a set of paths $\mathcal{T}$. In that case, we have for $t \in \mathcal{T}$:

1. Each path contains the requested access point $a_t = a_s$,
2. The total capacity of the paths equals the required capacity $\sum_{t \in \mathcal{T}} c_t = c_s$,

3. Each of the paths follows the latency constraint
   $\max_{t\in\mathcal{T}}(l_t) \leq l_s$,
4. The total cost of the paths follows the price budget $\sum_{t\in\mathcal{T}} c_t p_t \leq b_s$.

While serving service requests, each by its path or set of paths, it is required to follow the capacity constraints of all network components, i.e., to guarantee that for each component the sum of capacities allocated to paths containing the component is not larger than the capacity of the component.

We define the resource allocation and service provisioning time periods and their relation, illustrated in Figure 2.3.

Definition 2.5 (Resource allocation and service provisioning periods)**.** The resources that the customers lease are allocated for a fixed-length time period; this is the service provisioning period. Before this period, customers sequentially arrive with their service requests; that is what we call resource allocation period. Customers therefore book resources at the given cost at the selected providers during the resource allocation period for the service provisioning period.

Providers set price on their resources to be allocated for the service provisioning period. We assume that they maximize their expected profit for each service provisioning period. In order to do so they must take into account expected characteristics of demand (access points, capacity, latency, price budget) to appear during the resource allocation period. Furthermore, in any moment during the resource allocation period they have to consider the amount of resources that are already allocated for the given service provisioning period. They also have to compete with other providers' prices, we suppose they have complete information in this aspect.

Customers behave as followers, acting on the prices the providers have set. Customers arrive in a sequence one after the other, and seek a set of resources that can serve their requests. In case there is no suitable set of resources to serve a customer request, i.e., no option that satisfies its access point, capacity, latency and budget parameters, the customer is not served. If customers react to prices offered by resource providers, it is the typical setup of a Stackelberg game: leaders, in this game the resource providers, choose their strategies, i.e., set their prices, by taking into account the expected selfish decisions that the customers, followers in this case, will make.

After a customer finds an eligible resource set and allocates it, the providers reset their prices based on the available capacities, becoming the leaders again for the time the next customer comes. This way the Stackelberg game repeats itself although not identically in various stages as available capacities are decreasing from one stage to the subsequent one, while the allocation of capacities are cumulating until the point when the resource allocation period ends and the service provisioning period starts.

In the following we show how the followers can determine their best response strategies.

Any given customer with a service request $s = (a_s, c_s, l_s, b_s)$ during the resource allocation period would like to find a path (or a set of paths) with a total cost under

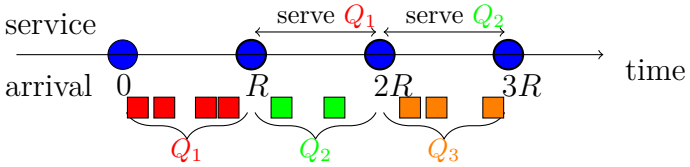


Figure 2.3: Resource allocation and service provisioning periods: In each time period (of a fixed length R) the served requests are those that arrived in the previous period.

its budget, while preserving the technical demands of s (on its access points, capacity and latency). The requested access point constrains the selection to be among a smaller resource subset. Similarly, the capacity requirement implies a subset of the network components from which the path can be found, following their available capacities. The impact of the latency and the budget constraints has a different flavor since these two metrics of a possible path (or a set of paths) are calculated in an aggregated way along the path resources. Accordingly, the path should be found while considering the following problem: given a network subset, find a path (or a set of paths) that preserves both the latency and budget constraints.

Theorem 2.2. *The problem of finding an eligible flow with at most b_s budget (or the cheapest eligible flow) is NP-hard if the network is an arbitrary graph.*

Proof. The problem of finding an eligible flow with at most b_s budget is the associated decision problem of finding the cheapest eligible flow problem. The latter problem is, on the other hand, equivalent to the *shortest weight-constrained path problem*, known to be NP-complete ([38], page 214). In this problem, given a graph in which each edge is associated with a length and a weight, it is required to determine whether there exists a path satisfying two upper bounds on its total length and weight. Even though in this problem values refer to links while in ours to nodes (latency and price), we can deduce the hardness for our problem. Note that polynomial-time algorithms exist for cases where all link weights are equal or alternatively all links have the same length. □

Due to the NP-hard nature of the eligible flow finding problem, we assume that the sequentially arriving customers apply heuristics when selecting the resource set to allocate. For this reason and because the service request parameters cannot be known in advance, in the next section we model the pricing game as a stochastic game, the customers being the randomness with a probabilistic behavior of setting request parameters and selecting paths.

2.3.2 Equilibrium prices in pricing games

In this section we list the analytical results of the evaluation of the pricing game. We set the stage for the stochastic game between the providers, given that customer requests are random in terms of parameters and selected paths. Then we examine three simple topologies and describe the equilibrium prices in those games.

As customers face an NP-hard problem with the eligible flow selection, we reduce the Stackelberg game, where providers are leaders, customers are followers, to a stochastic game among providers as players. In this stochastic game, the *state* is the available capacity level of resources at providers, the *actions* (or strategies) of the players are the prices that the providers set for themselves, and the *transition probability function* is determined by the probabilistic type of the customer that arrives next, and the path(s) it selects. Depending on which providers are chosen at resource allocation, which is a heuristic decision, providers' payoffs increase by a reward based on their given prices. Going forward we assume that customers allocate single paths (instead of flows), and thus we define the random variables of service requests as follows.

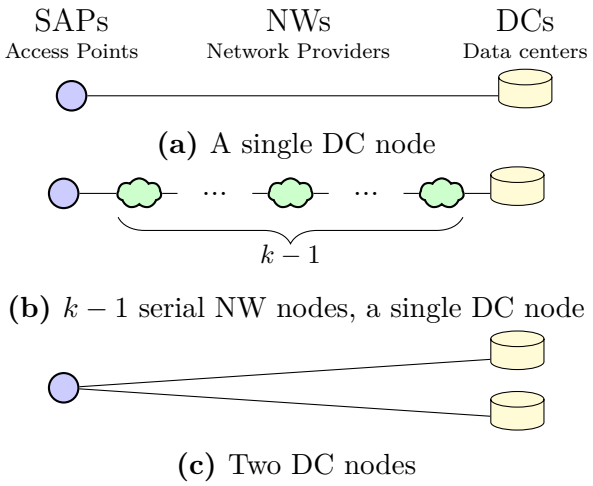
Definition 2.6 (Random variable of service request)**.** Let $\mathcal{S}$ denote the set of all possible requests and S the random variable of the next request. We define the random variable of service request by the tuple $S = (A, C, L, B)$, where we denote the random variables of the requested access point, the requested capacity, the latency constraint and the price budget by A, C, L and B, respectively. We suppose that each service s selects a single path during resource allocation from its access point a_s. Let $\mathcal{T}_s$ be the set of paths from a_s to $\mathcal{D}$ such that all of its elements satisfies C_s and L_s. Given $\mathcal{T}_s$, the tuple (T_s, B_s) fully defines the service request, where T_s is a random variable on $\mathcal{T}_s$. The path is chosen independently from the prices; if the selected path is costlier than the budget constraint, the request fails.

Feasible requests select such paths that satisfy capacity, latency and budget constraints. In such a setup, the central question is: what is the winning policy of the game players, i.e., the resource providers? In order to answer that, we seek the value of the game [92] for each player that gives the best-response actions in Markov-perfect equilibrium. Before doing so, we make an assumption on the memoryless arrival process.

Assumption 2.4. The number of requests during the resource allocation period follows geometric distribution with parameter q. Namely, each request is the last one with probability q and at least one other request follows with probability $1 - q$.

Without the loss of generality, we can suppose there is one element in $\mathcal{D}$, i.e., there is only one data center node in the topology under investigation, and its unit price is 0. If this is not true, i.e., if there are more elements in $\mathcal{D}$ or the one element's unit price is not 0, then for the analysis we turn the elements in $\mathcal{D}$ into elements in $\mathcal{N}$ with the same properties and insert an additional element into $\mathcal{D}$, connected to all original data center nodes. Assuming infinite capacity, zero delay and zero price for this one data center, we arrived to our initial assumption, keeping the original topology's characteristics intact from the analysis point of view.

For illustrative purposes, in the following we derive the equilibrium strategies for selected settings: the network topologies of providers represent both serial and parallel setups, latency constraints are supposed to be met for all eligible service requests, demanded capacity is fixed to one unit (large capacity services might be modeled by a number of unit-sized requests), and the price budget parameters of service requests are randomly drawn from uniform distributions. The goal of the analysis is



(a) A single DC node

(b) $k-1$ serial NW nodes, a single DC node

(c) Two DC nodes

Figure 2.4: Topologies with a single SAP, a 1 or 2 DCs and 0 or several NWs.

to formulate the best response strategies, i.e., the prices that providers set as their actions, through first determining the value of each game. We present 3 cases, as depicted in Figure 2.4. With these illustrative examples we present how the expected number of subsequent requests and the graph topology of interconnections determine the pricing strategies of the providers.

The example of Figure 2.4a shows how sequential allocations affect price, i.e., how to maximize revenue by setting a price with the expectation of future service requests. We suppose a widely used arrival process of Assumption 2.4: the probability of receiving a next service request does not diminish over time. Let V^1 denote the value of this game, although there is only one player here, hence the superscript, which is technically a decision problem. The recursive solution by capacity gives the following result for the provider with capacity c and price p.

Lemma 2.6. *If Assumption 2.4 holds, the value of the game is*

$$V^1(c,p) = \frac{\mathbb{P}(Z)\Big(p\mathbb{E}(C|Z) + (1-q)\mathbb{E}(W^1(c-C)|Z)\Big)}{q + \mathbb{P}(Z)(1-q)},$$

where $W^1(c)$ is the optimal value of the game for one player with c capacity, $\mathbb{P}(X)$ stands for probability of a stochastic event X, $\mathbb{E}(Y)$ denotes the expected value of a random variable Y, and $Z \overset{\text{def}}{=} (B \geq pC)$, i.e., the event of $B \geq pC$.

Proof. In general, the value of the game is

$$V^1(c,p) = \mathbb{P}(Z)(p\mathbb{E}(C|Z)+(1-q)\mathbb{E}(W^1(c-C)|Z))+(1-\mathbb{P}(Z))(1-q)V^1(c,p). \quad (2.3)$$

Z means that the request has enough budget to pay the price. When Z holds, the price p is paid, a capacity of $c-C$ remains for future requests. When Z does not hold, a value might be gained only from subsequent requests. □

Building on Lemma 2.6, and supposing unit-sized capacity requests and uniform distribution of budgets, we derive the closed-form formula of the equilibrium price.

Assumption 2.5. Service requests' capacity values are fixed $C \equiv 1$, budget constraint follows uniform distribution (denoted by $U(\min, \max)$), without loss of generality $B \sim U(0,1)$.

Lemma 2.7. *If Assumptions 2.4 and 2.5 hold, then the optimal price in the setup of Figure 2.4a is:*

$$p^* = \frac{1 - \sqrt{1 - (1-q)\left(1 - (1-q)qW^1(c-1)\right)}}{1-q}.$$

Proof. From Lemma 2.6 the value of the game for $c > 0$ is

$$V^1(c,p) = \frac{(1-p)\left(p + (1-q)W^1(c-1)\right)}{q + (1-p)(1-q)},$$

and $V^1(0,p) = 0$. In order to maximize the above function we get its derivative, which leads to the following equation for $p^* \in (0,1)$: $p^{*2}(1-q) - 2p^* + 1 - (1-q)qW^1(c-1) = 0$. From there, the statement is straightforward after some algebra. □

As for the equilibrium strategy, we know that $p* = \arg\max_p V^1(\hat{c}, p)$, so denoting $\max_p V^1(\hat{c}, p) = V^1(\hat{c}, p^*)$ as $V^1(\hat{c})$, we have $V^1(0) = 0$ and $V^1(1) = \frac{1}{(1+\sqrt{q})^2}$ with $p* = \frac{1}{1+\sqrt{q}}$. For further values of $\hat{c} > 1$, we solve the recursive formula $p^2(1-q) - 2p + (q-1)qV^1(\hat{c}-1) + 1 = 0$, $p \in (0,1)$ for p^* numerically. In Figure 2.5 we plot the equilibrium prices for different provider node capacities, i.e., $c \in [1, 30]$, and geometric distribution parameters, i.e., $q = \{0.1, 0.2\}$. First, we can see that higher chances for the arrival of subsequent requests (low values of q) motivates the resource provider to demand a higher price. On the other hand, having a larger capacity for the resource, motivates it not to be strict and to accept requests with lower budgets. Inspired by the observations of the numerical solutions depicted in Figure 2.5, we derive the equilibrium price of the game analytically for $c = \infty$: when serving a request has no negative impact on the remaining capacity, the resource provider tries to maximize the value obtained from every request independently. We prove that the price should not be too low or high in terms of the budget distribution.

Lemma 2.8. *If Assumption 2.5 holds and the provider's capacity is infinite, i.e., $c = \infty$, the equilibrium price is $\frac{1}{2}$.*

Proof. With $c = \infty$ the game reduces to focusing on a single request. Then similarly to Lemma 2.6 the value of one stage is: $V^1(\infty, p) = p\mathbb{P}(B \geq p) = p(1-p) \to p^* = \frac{1}{2}$. □

Namely, with infinite capacity, providers maximize the profit from each request, independently. While setting a high price can increase the profit when taken, it has

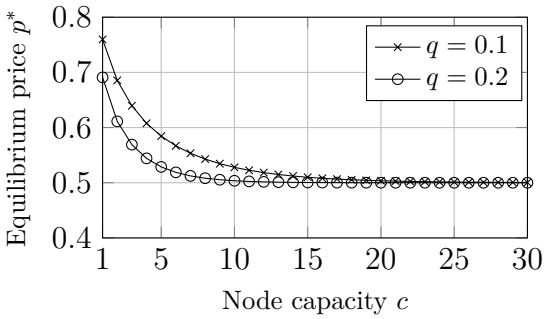


Figure 2.5: Equilibrium prices for various c, q values in the single NW topology.

lower chances to be a part of a selected path. With a uniform distribution of the budget $B \sim U(0, 1)$, we get that the equilibrium price is 0.5.

Here we analyze the game in which k network providers are adjacent to each other, as depicted at Figure 2.4b. Similarly to the previous case, we can determine the equilibrium prices for the restricted case of uniformly distributed price budgets and fix capacities in the service requests.

Lemma 2.9. *If Assumptions 2.4 and 2.5 hold, and the number of network providers is k in the serial setup, then for the Nash-equilibrium we have the following equation:* $0 = k^2(1-q)p^{*2} - p^*(2k - q(k-1)) + 1 - q(1-q)W^s(c-1)$.

Proof. Similarly to Lemma 2.6 the value of the game is:

$$V_i^s(c, \mathbf{p}) = \frac{(1 - \sum p_j)(p_i + (1-q)W_i^s(c-1))}{q + (1 - \sum p_j)(1-q)},$$

where V_i^s is the game value for provider i (s superscript as in serial), and $\mathbf{p} = (p_1, p_2, \ldots, p_k)$. The numerical analysis of the equilibrium prices is based on the observation that the network provider nodes have symmetrical role, so we suppose $p_i^* = p^* \ \forall i$. Then we have the above equation for p^*. □

In Figure 2.6 we plot the numerical values of prices calculated with the formula in Lemma 2.9, now in the function of the number of providers ($k = [1, 10]$), and their capacities ($c = [1, 20]$). The expected number of requests is 10, i.e., $q = 0.1$. Results show that prices are inversely proportional to the number of players, and converge fast as the values of capacities leave the range of under-provisioning. Again, inspired by these results, from Lemma 2.9 we have the following direct consequence for the case where network capacities are unlimited. For this latter case, a solid blue line depicts the numerical values in Figure 2.6.

Lemma 2.10. *If Assumption 2.5 holds and the providers' capacity is infinite, i.e., $c_i = \infty$, then the equilibrium price and ratio to the the price of anarchy for k nodes are:*

$$p^* = \frac{1}{k+1}, \ PoA = \frac{(k+1)^2}{4k}$$

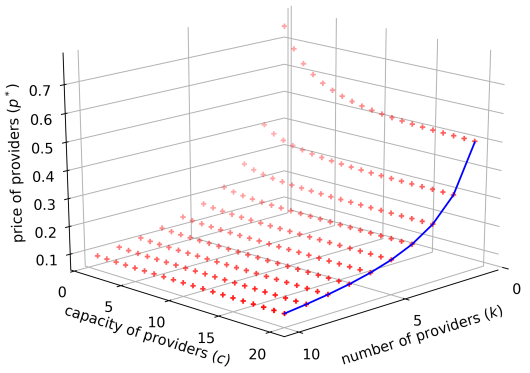


Figure 2.6: Equilibrium price $p^*(c)$ for $k \in [1, 10]$ providers and arrival q=0.1. Markers are used to describe the price for a finite capacity $c \in [1, 20]$ and a solid line for an infinite capacity $c = \infty$.

Proof. To derive equilibrium prices for infinite capacities it is enough to focus on only one request. The value of a request is $p_i(1-\sum p_j)$. It is maximal if $2p_i = 1-\sum_{j\neq i} p_j$, and in equilibrium p_i^* are the same, so $p^* = \frac{1}{k+1}$. For k providers the price of the path is $\frac{k}{k+1}$, so the value for one provider is $p^*\mathbb{P}(B > \frac{k}{k+1}) = \frac{1}{k+1}\frac{1}{k+1}$. The game value for one request is $W^s(\infty) = \frac{1}{(k+1)^2}$, and for the whole game it is $V^s(\infty) = \frac{1}{1-q}W^s(\infty)$. If providers maximized the sum of their game values, the sum of probability would be $\frac{1}{2}$, which would lead to $p = \frac{1}{2k}$ and for the maximal value it would give $\frac{1}{4k}$. □

The inter-dependent providers drive their prices to an inversely proportional drop, and to a slight increase in the total price for the customer. The price of anarchy grows linearly with the number of providers: the longer the chain of providers, the further they shift away from Pareto optimum.

After the examples of one, or more players on the same path, optimizing their expected total income over the resource allocation period, now we turn to the simplest competitive game of "substitute" providers. In this setting two DC providers are both connected to the SAP in parallel, posing direct competition to each other. This is the scenario depicted in Figure 2.4c. Let V^p denote the value of this so-called parallel game.

Now we give the general formula for the value of this game, assuming that the arrival process of service requests is memoryless. If Assumption 2.4 holds, then:

$$\begin{aligned} V_1^p(\mathbf{c},\mathbf{p}) = \frac{\mathbb{P}(S_1 \wedge S_2)}{2} p_1 \mathbb{E}(C|S_1 \wedge S_2) + \mathbb{P}(S_1 \wedge \overline{S_2}) p_1 \mathbb{E}(C|S_1 \wedge \overline{S_2}) \\ + (1-q)\Big[\mathbb{P}(S_1 \wedge S_2)\frac{1}{2}\mathbb{E}(V_1^p(c_1 - C, \hat{c}_2) + V_1^p(c_1, c_2 - C)|S_1 \wedge S_2) \\ + \mathbb{P}(S_1 \wedge \overline{S_2})\ \mathbb{E}(V_1^p(c_1 - C, c_2)|S_1 \wedge \overline{S_2}) + \mathbb{P}(\overline{S_1} \wedge S_2)\mathbb{E}(V_1^p(c_1, c_2 - C)|\overline{S_1} \wedge S_2) \\ + \mathbb{P}(\overline{S_1} \wedge \overline{S_2}) V_1^p(\mathbf{c},\mathbf{p})\Big], \end{aligned}$$

where $S_i \stackrel{\text{def}}{=} (C \leq c_i) \wedge (p_i \leq B)$, $\mathbf{c} = (c_1, c_2)$, $\mathbf{p} = (p_1, p_2)$, and $\bar{S}$ denotes the complement of S.

In case of uniform budgets and unit-sized jobs, the price strategies are delivered by the following formula.

Lemma 2.11. *If Assumptions 2.4 and 2.5 hold, in the two-player parallel game the solution of these equations provide the equilibrium prices for the 2 players, denoted by i and $-i$:*

$$0 = p_i^{*2}(1-q) - p_i^*(4 - 2p_{-i}^*(1-q)) + 2 - (1-q)(2W^p(c_i - 1, c_{-i}) + p_{-i}) \\ +(1-q)^2(p_{-i}(W^p(c_i-1, c_{-i}) - W^p(c_i, c_{-i}-1)) + (1-q)^2(W^p(c_i-1, c_{-i}) + W^p(c_i, c_{-i}-1)). \tag{2.4}$$

Proof. Similarly to the previous cases, the value of the game for the first provider can be written as:

$$V_1^p(\mathbf{c}, \mathbf{p}) = \frac{(1-q)(1-p_1)W^p(c_1 - 1, c_2)}{2 - (1-q)(p_1 + p_2)} + \frac{(1-q)(1-p_2)W^p(c_1, c_2 - 1) + p_1(1-p_1)}{2 - (1-q)(p_1 + p_2)} \tag{2.5}$$

By deriving it according to p_1, we get the equation above. □

If provider capacities are infinite then the game is again reducible to the case of only one request. In that case the providers' game values are independent from each other, which means the game reduces to the single node case, where the optimal price is $\frac{1}{2}$. The numerical solution of Lemma 2.11 depicted in Figure 2.7 shows in the case of $q = 0.1$ how the equilibrium price converges to $\frac{1}{2}$ as the capacities go to infinite. It also means that the price of anarchy converges to 1 for this topology, as in the single node case. The prices in case of infinite provider capacities are depicted again by solid blue lines in Figure 2.7.

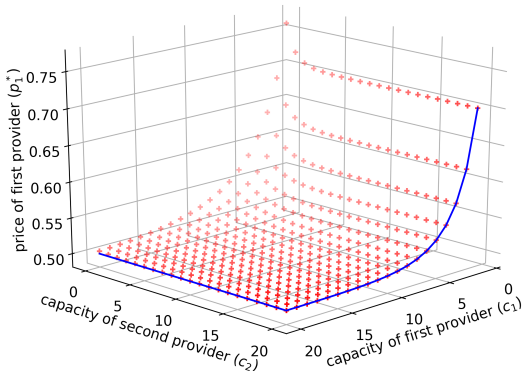


Figure 2.7: Equilibrium price $p_1^*(c_1, c_2)$ for provider capacities $c_1, c_2 \in [0, 20]$ and arrival $q = 0.1$. Markers are used to describe the price for finite capacities $c_i \in [0, 20]$ and a solid line for infinite capacities $c_i = \infty$.

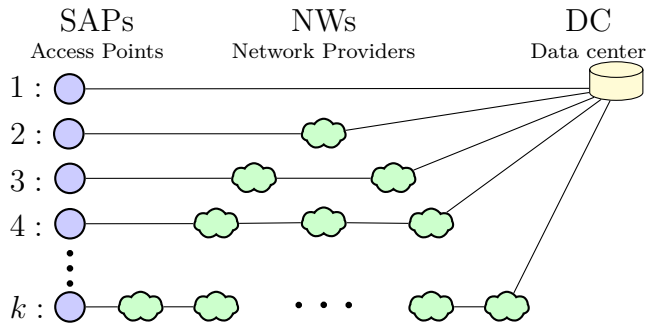


Figure 2.8: Parallel paths with serial NWs.

In the next section we investigate such cases in which the providers' capacity can be treated as infinite, compared to that of a single request.

2.3.3 Analysis of infinite-capacity games

Hindered by the complexity of the game analysis, and building on the observation of fast price convergence in function of provider capacities, in this section we relax the capacity constraints of service requests. By doing so, first we derive the equilibrium prices for an artificial, but more complex topology than the previous ones; second, we formulate equations that provide equilibrium price values for general topologies.

Assumption 2.6. Network and compute providers have sufficient capacity to accommodate service requests: the set of available paths of a given request is not limited by capacity constraints.

Here we show the equilibrium prices for the topology shown in Figure 2.8: in this artificial topology we draw k chains of NWs with different lengths that connect k SAPs to the same DC. The equilibrium prices are given in the next proposition.

Lemma 2.12. *If Assumptions 2.5 and 2.6 hold, the topology is as it is given in Figure 2.8 and the requests contain uniformly the k SAPs, then the equilibrium prices are:*

$$p_D^* = \frac{H_k}{k + H_k},\ p_i^* = \frac{k/i}{k + H_k},\ PoA \sim \frac{k + H_k}{4H_k}$$

where p_D^* is the price of the DC, p_i^* is the price of nodes in chain with $i-1$ NWs and $H_k = \sum 1/j \sim \ln k$.

Proof. Assuming that the providers in the same chain have the same optimal price, from the derivatives of the value functions we have the following equation for p_i $i \in [2, \ldots, k]$ and p_D:

$$p_i = \frac{1 - p_D}{i},\ p_D = \frac{k - \sum(i-1)p_i}{2k}.$$

□

On one hand, $p_D \sim \frac{\sum 1/i}{k}$, i.e., the data center's price is proportional to the average of the path length reciprocals. On the other hand, the even more important result

from this topology is that if k is large enough, then $p_i^* \sim 1/(i+1)$, where i is the length of the path that contains the node. With this observation in hand, we now turn to another topology in which network provider nodes are situated on shared paths to the same DC.

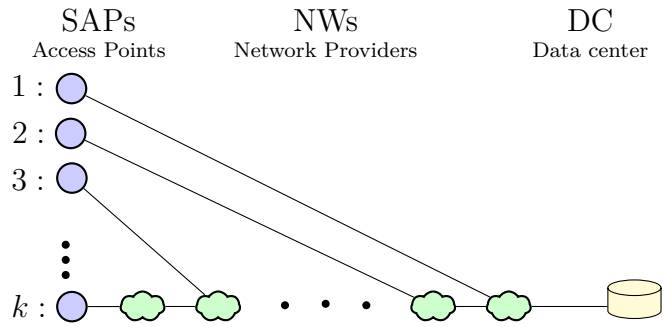


Figure 2.9: The pine-tree topology.

In this second case the topology, depicted in Figure 2.9, contains a chain of k providers and each of them has an SAP, and there is one DC at the end of the chain.

Lemma 2.13. *If the topology is the same as in Figure 2.9, and the requests arrive uniformly from the k SAPs, then the Nash-equilibrium is:*

$$p_i^* = \frac{e_i(i+1)(i+2)\dots k}{(e_i+i)(e_{i+1}+i+1)\dots(e_k+k)},$$

where $e_1 = 1$ and $e_{i+1} = \frac{i+(i+1)e_i}{i+e_i}$. NW nodes are enumerated based on the number assigned to their connected SAP as depicted in Figure 2.9

Proof. As $V_i(\mathbf{p}) = \frac{1}{k} p_i \sum_{l=1}^{i} \left(1 - \sum_l^k p_j\right)$, considering the equations $\frac{\partial}{\partial p_i} V_i = 0$, we get the following equations: $p_i = \frac{e_i}{i+e_i}\left(1 - \sum_{i+1}^k p_j\right)$, which gives the above solution. □

Figure 2.10 shows that in the numerical evaluation of the above formula the price monotonously decreases NW node-by-NW node going from the DC towards the edge, i.e., providing transport for fewer and fewer SAPs. Calculations were run for different values of k: $[2, 10]$, lines represent the simulations, the more nodes are involved, lower the price curve is located on the plot.

In the following we formulate the equations that give equilibrium prices of the providers for general topologies. If t is a requested path then let us denote with $\bar{t}$ the set of vertices on t except the service access point.

Lemma 2.14. *If Assumptions 2.5 and 2.6 hold we have the following equations for the prices for all $x \in G \setminus \mathcal{A}$:*

$$0 = \sum_{s\in\mathcal{S}} \sum_{x\in t, t\in\mathcal{T}_s} \mathbb{P}(S=s, T_s=t) \left[\left(1 - F_{B_s}\left(\sum_{y\in\bar{t}} p_y\right)\right) - f_{B_s}\left(\sum_{y\in\bar{t}} p_y\right) p_x \right] \qquad (2.6)$$

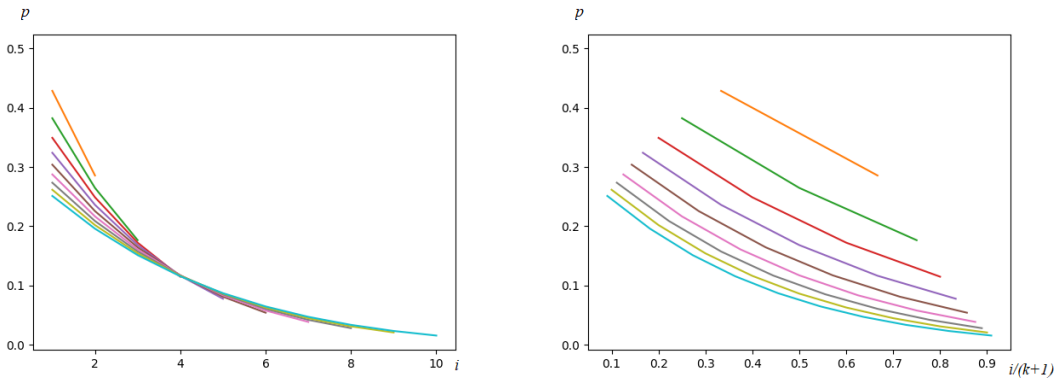


Figure 2.10: Equilibrium price for NW nodes according to their distance from the DC in the top (x-axis: i, y-axis: p_i^*), and the same distance-dependent prices, but the sequential number of nodes are normalized by $k+1$ in the bottom (x-axis: $i/(k+1)$, y-axis: p_i^*) figure. One line corresponds to one numerical evaluation of formula in Lemma 2.13 for a given k value.

Proof. For every $x \in G \setminus \mathcal{A}$, V_x denotes the value of the game and p_x the unit price of resource provider x respectively.

$$V_x = \sum_{s \in \mathcal{S}, t \in \mathcal{T}_s, x \in t} \mathbb{P}(S = s, T_s = t) \left(1 - F_{B_s}\left(\sum_{y \in \bar{t}} p_y\right)\right) p_x$$

If $\frac{\partial}{\partial p_x} V_x = 0$, then we get the above equations. □

Finally, we make an important statement for cases in which customers' path selection is not random, instead determined by Assumption 2.7 below.

Since finding the cheapest eligible path is NP-hard (see Theorem 2.2) we suppose that customers do not solve this problem. Instead we propose for the customers to use the following heuristics to select a path in polynomial time: choose the eligible path with minimum number of hops. It is well known that the *shortest weight-constrained path problem* is polynomial if the link weights are equal, and in this case they are all ones. The reason to apply this heuristics is that as we see in Lemma 2.12 the shorter a path (in hops), the cheaper. We believe this is true in general situations, not only in the case of Lemma 2.12.

Assumption 2.7. Let all $a \in \mathcal{A}$ deterministically choose the shortest path (in terms of hops) which is eligible, t_a, to the closest $d \in \mathcal{D}$. In this case $\mathcal{S} = \{s_a | a \in \mathcal{A}, s_a = (t_a, B_a)\}$.

Assuming this shortest path selection policy, Lemma 2.14 directly leads to the next consequences.

Corollary 2.1. *If Assumptions 2.5, 2.6 and 2.7 hold we have the following linear equation system on the probability of a provider being on a path selected by a service*

request. The equilibrium prices can be calculated from these equations.

$$\mathbb{P}(x \in \overline{T_S}) = \sum_{a \text{ if } x \in \overline{t_a}} \mathbb{P}(S = s_a) \left(p_x + \sum_{y \in \overline{t_a}} p_y \right),$$

$\forall x \in G \setminus \mathcal{A}$.

Corollary 2.2. *If Assumptions 2.5, 2.6 and 2.7 hold all used SAP - DC paths have the same length, and the number of non-AP nodes on the path are k ($\forall a \in \mathcal{A}, |\overline{t_a}| = k$), then the equilibrium prices are $p_x = \frac{1}{k+1}$ for all $x \in G \setminus \mathcal{A}$.*

Moreover, we can also deduce the price of anarchy.

Corollary 2.3. *For the case Assumptions 2.5, 2.6 and 2.7 hold, the price of anarchy can be calculated from the equilibrium prices p_x for $x \in G \setminus \mathcal{A}$ by the following way:*

$$PoA = \frac{1}{4 \sum_{a \in \mathcal{A}} \mathbb{P}(S = s_a)(1 - \sum_{y \in \overline{t_a}} p_y)(\sum_{y \in \overline{t_a}} p_y)}.$$

Proof. The sum of the game values can be maximized by the following way: DCs set the price to $\frac{1}{2}$, NWs set the price to 0. In this case the maximum is $\frac{1}{4}$. □

Corollary 2.1 provides an equation system that gives the equilibrium prices of any given topology. Corollary 2.2's statement is similar to those that we made for special cases in the previous section, but in this case it stands for general topologies. The importance of this statement is supported by the fact that in realistic settings SAPs are typically at the edge of the network, while DCs are in the center, so the length of most of the paths equals to the radius of the network. Finally, Corollary 2.3 provides the formula for the price of anarchy. In cases when paths typically contain 4 providers, $PoA \sim 1.5$, if they contain 10 providers the $PoA \sim 3$.

2.4 Related work

Here we summarize related work that connects to our domain of research. We group the collected research papers into three categories: resource pricing in the cloud, network formation games, Internet connectivity prices and agreements.

2.4.1 Resource pricing in the cloud

Managing network and cloud resources together in cloud networking has many challenges. [74] reviews pricing models for resource management in this scope. Most of the collected works therein propose the application of dynamic pricing, as it increases seller's profit when two product characteristics co-exist: first, the product expires at a point in time, second, capacity is fixed and it is costly to be augmented.

Both are true for the market of leasing data center resources. The term cloud networking is perceived in a multi-administration situation in which network and cloud domains interface with one another.

When goods are sold with dynamic pricing, either markets generate the prices, e.g., in auctions, or sellers apply pricing schemes and market mechanisms to adapt dynamically to the demand. A real-world example for the former is a spot market where the price is given by the intersection point of demand and supply, as for Amazon EC2 Spot Instances [8]. In the latter case, there are various techniques for maximizing seller surplus. In [136] the authors proposed strategies that a provider can use to improve revenue, including resource throttling. [17] presents an algorithm for the admission and allocation of network slice requests that maximizes the infrastructure provider's revenue. Related to pricing aspects of 5G network slices, in [17] the authors proposed a model for network slicing under the control of a single infrastructure provider: they introduced an analytical model for admission control of slicing-capable 5G networks. Furthermore, they also proposed a method for optimizing the revenue of the infrastructure provider. The authors of [20] applied the share-constrained proportional allocation mechanism for network slicing. The research in [97] focuses on mobile cloud computing, i.e., the integration of cloud computing into a mobile ecosystem, thus enabling on-demand and elastic resource utilization for mobile users. The article, however, emphasized that further work was required to explore the trade-off between the benefits of uploading tasks and the price that mobile users must pay for the same. In [127], the authors tackled a problem seemingly similar to ours with the same tool set: resource allocation for network slices among multi-tier stakeholders in a Stackelberg game. The problem they modeled, however, covered wireless channel allocation of access providers, i.e., leaders, and user equipment subscriptions, i.e., followers. The payoff models were built such that convex optimization problems arose, i.e., followers set their demand independently from each other, but in response to the prices advertised by the leaders; leaders determined their prices in an iterative process by giving best response to others' prices, but independent of the price to pay towards the backhaul providers. The latter were also part of the analysis, i.e., determining the price of the bandwidth to sell to access providers, but the backhaul providers were not part of the Stackelberg game.

The term cloud networking is understood in a multi-administrative domain scenario in which network and data center domains interact with each other. While major effort lies within the design and evaluation of NFV management and orchestration solutions [32], interestingly there are no proposals for pricing or including prices in orchestration methods among standards [33], and to the best of our knowledge only a few research works have addressed these problems so far. In the multi-provider NFVIaaS market however, for the implementation of the visions of 5G it is necessary to price cloud and network resources in a market for many providers and customers.

The following related works made modeling and evaluation steps in this direction. In [64] the authors analyzed pricing schemes for the joint provisioning of radio access capacity and mobile edge computing services in a multi-tenant Radio Access Network. A usage-based dynamic pricing scheme applied for SFCs is presented in [87]: the authors derive the prices, based on the actual utilization and on historic

data, that maximize the potential income of the provider. In all of this related work, the optimal pricing scheme for compute and/or network resources was sought, mostly in a single-provider multi-user setting. In contrast to these work, our endeavor is to model the business costs of interaction between multiple providers and customers.

2.4.2 Network formation models

The second area of the related work is best described by network formation models, inspired by the stability of the Internet despite its sheer size. Large computer networks, such as the Internet, are built, operated, and used by a large number of diverse and competitive entities. In light of these competing forces, it is surprising how efficient these networks are. An exciting challenge in the area of algorithmic game theory is to understand the success of these networks in game theoretic terms: network formation games are widely used to investigate the principles of interaction that lead selfish participants to form efficient networks.

Fabrikant et al. presented the network formation game that modeled the dynamic building process of networks by economically selfish nodes without any central coordination [34]. In their model, nodes pay for the connections that they set up, and pull advantage from the short routes to all fellow nodes. The authors examined the Nash equilibria of the game, and inferred results about the "price of anarchy", i.e., the overall loss due to the absence of coordination.

Corbo et al. contemplated a network formation game where connections required the assent of the two nodes and were negotiated reciprocally [25]. The authors contrasted these networks with those produced by the prior model of [34] in which connections were created unilaterally. Their observations exhibited that the average price of anarchy was better in the reciprocal network formation game than in the one-sided game for little connection costs, however worse as connections become more costly. In the book chapter [116] the authors investigated various different network formation games regarding the loss that comes from game theoretical selfishness.

Another work that handles reciprocally concurred agreements was introduced in [13]: cost was caused to a node from four sources: (1) routing traffic; (2) keeping up connections to different nodes; (3) getting disconnected from nodes it wished to reach; and (4) installments made to different nodes. The authors studied the game in context of the idea of pairwise stability. The distinction contrasted with our work is that our model records for the "routing" term as income source, rather than making it a cost-expanding term.

All these network formation models and the games defined on them considered a flat, single-layer topology in which nodes, i.e., network providers, are homogeneous. In this aspect, our assumption about the players is different: as in the related work on the economics of the Internet, discussed in the following, we consider a tiered setup of nodes in which upper layer network providers are significantly larger in terms of infrastructure and end user base than lower layer network providers. This context makes the network formation game entirely different.

2.4.3 Internet economics

The authors of [108] focused on the Internet topology as they analyzed Internet Service Provider (ISP) interactions at different levels: (1) when they compete directly for customers, (2) when they belong to different levels of the hierarchy, and (3) when they try to bypass a section of the hierarchy. The paper examined the existence of equilibrium strategies through a Stackelberg game [26] along the ISPs' tiered hierarchy. The authors derived conclusions on the possible evolution of the Internet topology focusing on local ISPs, anticipating that ISPs in each region would set similar prices and provide similar QoS levels.

Dhamdhere et al. investigated how the Internet ecosystem evolved from a multi-tier hierarchy, built mostly with transit (customer-provider) links, to a dense mesh, formed with mostly peering links [28]. They studied this evolutionary transition with an agent-based network formation model that captured key aspects of the inter-domain ecosystem, e.g., traffic flow and routing, provider and peer selection strategies, geographical constraints, and the economics of transit and peering interconnections. Their model predicted several differences between the *Hierarchical Internet* and the *Flat Internet* in terms of topological structure, path lengths, and the profitability of transit providers.

The same authors published an agent-based network formation model for the Internet at the Autonomous System (AS) level in [69]: ASes act in a myopic and decentralized manner to optimize a cost-related fitness function, capturing key factors that affect the network formation dynamics, such as highly skewed traffic matrix, policy-based routing, geographic co-location constraints, and the costs of transit/peering agreements. As opposed to analytic game-theoretic models, which focus on proving the existence of equilibria, this was a computational model that simulated the network formation process and allowed to actually compute distinct equilibria (i.e., networks) and to also examine the behavior of sample paths that do not converge. They found that such oscillatory sample paths occur in about 10% of the runs, and they always involved Tier-1 ASes, resembling the Tier-1 peering disputes often seen in practice. In another work [70], the same authors investigated why many transit providers apply open peering strategy. They also examined the impact of an *Open peering* variant that requires some coordination among providers.

These selected papers from the body of research on Internet economics focused on the hidden reasons of the Internet topology evolution, specifically the decade-long investigation on transit vs. peering relations in the Internet. In the late 2000s, peering links between ISPs started to form in great numbers. This phenomenon inspired the researchers, and several of them applied game theoretical methodologies to model and explain this prominent trend. Although the methodology we apply is similar, our focus is not on the exchanged traffic between ISPs, rather on the multi-provider resource slice creation between infrastructure providers that might operate at remote geographic location far from each other. However, the sense of end-to-end connectivity is also found in our research domain, only not in the classical user plane sense, rather as business relationships.

Chapter 3

Resource provisioning within cloud-based systems

3.1 Introduction

Today's services strive for worldwide availability and geographic reach might be even more crucial in the future. In order for a system to meet all the requirements, e.g., low latency and high availability practically everywhere, it should have tens of thousands of computing nodes geographically distributed and fully connected to serve all the clients. Generally, we can state that managing such a huge infrastructure is far from trivial, and exacerbated by the geographical spread. Answering simple questions, like, how do we measure network characteristics effectively, how can we react to topology changes, do become more and more difficult. Besides availability, reaching high reliability is also challenging: service providers must ensure that they can respond to different failures, so their users will not be affected by a service outage for a long time.

In this chapter we give potential answers to these challenges and provide a conceptual solution for the fulfillment of strict time criteria of future applications. We propose our advanced edge-scheduler that takes into account the underlying network latency, and the applications' latency requirement in the scheduling decisions about the application components. Our backup resource multiplexing technique provides high reliability for the applications with aware of latency requirements by carefully provisioning resources for this aim. The system works on large scale with huge number of worker nodes and service requests thanks to our dynamic clustering method that can also organize a federated system dynamically.

In order to sustain QoS for the cloud-deployed application's users, cloud resources are continuously adapted to fit the demand, e.g., the time varying amount of client requests hitting the application ingress. When incoming demand is hectic, SLA constraints are strict, and the budget for cloud resources is tight, one must apply a sophisticated auto-scaler logic. This is the case in edge cloud setups that are characterized by limited infrastructure capacity and a relatively small number of clients, the requests of which are not multiplexed into a stable demand, while typical edge applications promise strict SLAs, e.g., low latency. In this chapter we also

provide an auto-scaler engine that adapts scaling decisions to the actual demand, not constrained by limited information, or by the rigidity of the scaling logic.

When software is packaged as a set of microservices, its modularity is high, both in terms of development and operation. As a consequence, extra delay might be introduced in the cloud application end-to-end latency due to the following reasons: i) such distributed applications have to count with an inter-component invocation delay, and ii) this is exacerbated by the inter-node network latency of the data center, if application components are orchestrated to run on different compute nodes [45]. However, if modules are packaged together and therefore scaled together, scale-out actions may lead to unnecessary resource, e.g., memory, consumption, while the co-location of application components within the cloud results in lower operational delays, hence better QoS for the application user. In this chapter we also propose a model to define the trade-off between response time performance (i.e., latency) vs. cloud resource footprint when it comes to the decision about designing, packaging and deploying a cloud native application.

The rest of the chapter is organized as follows. In Section 3.2 we introduce our model for ensuring high reliability and ultra-low latency with economical edge resource provisioning: we show our proposed scheduler solution that is based on an advanced heuristic scheduling algorithm, which dynamically handles incoming events of a geographically widespread virtual infrastructure, supporting several latency critical 5G applications. In Section 3.3 we present the operation of our re-scheduler that further decreases the provisioned resources in our system periodically, in an offline orchestration operation. We address scalability and present our edge node clustering solution based on network delay in Section 3.4. In Section 3.5 we evaluate auto-scaling methods analytically with a queuing theory-based model, and then inspired by the vast body of research performed in time series analysis and scaling decision-making in various systems ranging from power grids to cloud services, we propose a Machine Learning (ML)-based scaling method in order to capitalize on such well-known phenomena as daily and weekly profiles, and changing variability of request intensity throughout the day. Finally, in Section 3.6 we analyze the effects of various packaging options of cloud-native applications.

3.2 Scalable and economical edge scheduling for latency-, and operation-critical applications

Our proposed concept turns a virtual infrastructure scheduler into a manager of geographically widespread infrastructure. It is built on two advanced scheduling algorithms that support latency critical applications. In this section we introduce our proposition for reserving backup resources, and we build a model for describing the problem of minimizing the amount of those while conveniently providing reliability for delay-critical applications on top of them.

In the architecture of the system we call the physical entities with computational resources, processor, memory, network bandwidth, as nodes. In this sense, a node can represent a single server at the edge of the network, or an abstraction of an

entire cloud data center. Since our system considers network latency in every aspect of application deployment, we use a delay matrix: the values in the delay matrix represent the smallest delay value between each node pair. The deployable units of application components are called as Pods. The users can define delay criterion for each latency critical Pod which gives the maximum network latency that is tolerated by the application from an arbitrary point defined by the application provider, which we call *origin*.

Definition 3.1 (Pod deployment request)**.** The application provider submits its requests to deploy each and every Pod in the system along with the Pod's origin and with the respective radius. The origin of a Pod can be defined either as one of the nodes (that is close to the location of the users that the Pod will serve), or as another Pod deployed previously in the system (with which affinity is required for the Pod currently being deployed).

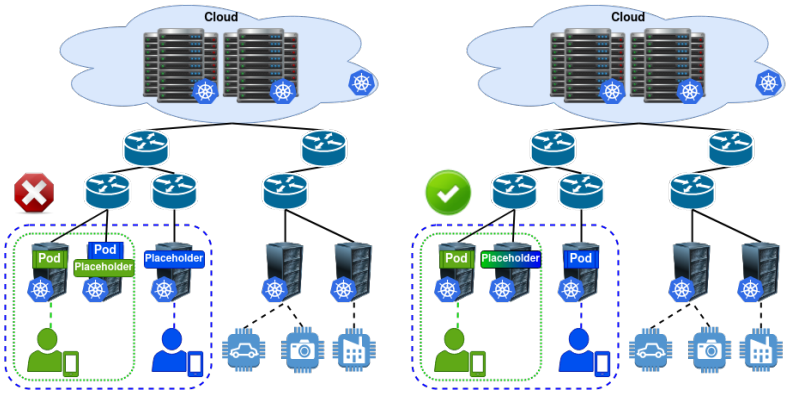


Figure 3.1: Proposed system architecture design for edge computing

To provide high reliability for the applications, we provision backup compute resources on edge nodes, which we call *placeholders*. We prepare for only one node failure at a given moment, so we dimension the placeholders for the maximum number of Pods on any node to fail at once. Therefore, each placeholder has a resource demand depending on two factors: i) how many Pods it supports simultaneously; ii) how the backed up Pods are distributed among the nodes. A placeholder's size is not necessarily equal to the sum of the supported Pods' sizes: it can be less if the supported Pods are placed on different nodes. A Pod's placeholder must be assigned to a node that differs from the host of the Pod, and the placeholder must also fulfill the Pod's latency requirement.

Since edge nodes are prone to failures and we strive to ensure high reliability for all the applications, we want to make sure if a single node failure occurs, placeholders in the system have enough reserved resource to restart all Pods of the failed node. We consider that the resources on edge nodes are expensive, since edge nodes have limited resource capacity compared to the large data centers. The Pods' have computational characteristics that our system needs to ensure, i.e., processor, memory, network bandwidth. Regarding of these two properties, one of our main goals is to minimize the resources reserved for placeholders in the system.

An example view of our system architecture with two simple scheduling results is presented in Figure 3.1 with a central cloud, and several edge nodes. On the left side of Figure 3.1, a non-optimal scheduling example is presented. In this case, the amount of backup resource (placeholder) reservation is greater than what the optimal solution would need. The result of how an advanced scheduler would deploy both the Pods and the placeholders can be seen on the right side. Since both of the Pods have common servers in their latency radius, their placeholders can be multiplexed in order to decrease the provisioned extra resources but still high reliability is ensured for the Pods.

The architecture of our edge-scheduler with some mandatory operations of each component is visualized in Figure 3.2. The main components in our edge-scheduler are the Monitoring, Event handler, Clustering, Scheduler, and Re-scheduler.

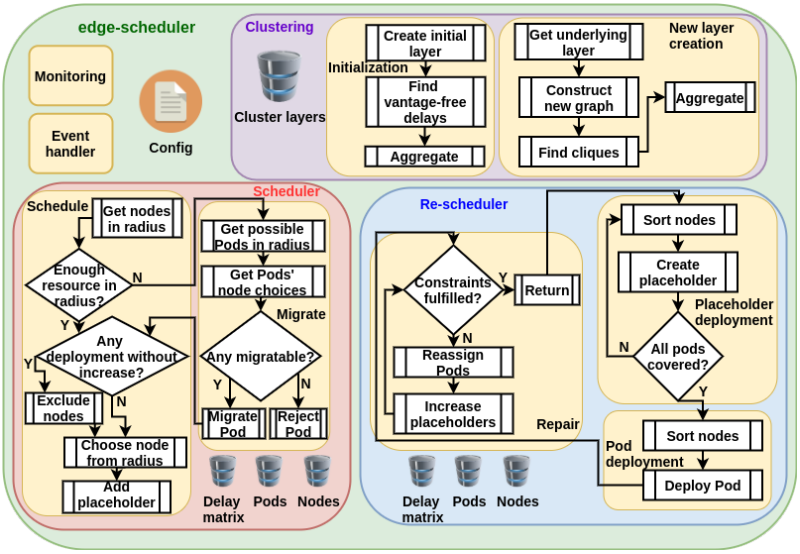


Figure 3.2: The components of our edge-scheduler

Our online scheduler component is in charge of deploying the incoming Pod requests on the fly, with the awareness of their delay and computational requirements, and also of deploying respective placeholders for ensuring high reliability. It works in polynomial time and its approximation ratio is 3 in terms of total amount of placeholder resources sacrificed for guaranteeing high reliability against single node failures.

The scheduler works in an online manner, it processes the users' application requests one-by-one at the time of their submission. The major steps of scheduling are showcased in the schedule box in Figure 3.2 and Algorithm 1.

Algorithm 1: Online schedule algorithm

```
   Data: pod, nodes
   Result: pod.node and pod.placeholder are set
 1 if pod.reschedule then
 2 |   bind(pod, pod.target);
 3 |   pod.reschedule ← False;
 4 end
 5 if pod.placeholder then
 6 |   patch(pod, pod.placeholder);
 7 else
 8 |   nodes_in_radius ← nodes.filter(pod.origin, pod.delay) ;
 9 |   suitable_nodes ← nodes_in_radius.filter(pod.capacity);
10 |   while suitable_nodes.size = 0 do
11 |   |   nodes_in_radius ← find_and_migrate_pods(nodes_in_radius) ;
12 |   |   suitable_nodes ← nodes_in_radius.filter(pod.capacity);
13 |   end
14 |   if suitable_nodes.size > 1 then
15 |   |   hosting_nodes, placeholder ← find_placeholder(suitable_nodes,
   |   |     pod) ;
16 |   else
17 |   |   No placeholder will be assigned to this Pod! ;
18 |   end
19 |   if hosting_nodes.size > 0 then
20 |   |   chosen_node ← min_utilized(hosting_nodes) ;
21 |   else
22 |   |   chosen_node ← min_utilized(suitable_nodes) ;
23 |   end
24 |   bind(pod, chosen_node);
25 |   if placeholder then
26 |   |   patch(pod, placeholder) ;
27 |   else
28 |   |   create_placeholder(suitable_nodes, pod) ;
29 |   end
30 end
```

Since our scheduler must give a solution that meets the delay requirements, the scheduling starts with the identification of the options that the Pod's requirements allow. More precisely, it pinpoints the nodes that are in the radius of the given origin (Line 8 of Algorithm 1). In case of the origin is a Pod, the algorithm defines its current host and gets the nodes around that. It is possible that none of the nodes in the radius have enough computational resource for the new application, i.e., processor, memory, network bandwidth. If none of the listed nodes have enough computational resources, our algorithm tries to migrate Pods from their current hosts to somewhere else, in order to free up some resource for the actual request (Line 11 of Algorithm 1).

During the Pod scheduling, we have to keep in mind the scarcity of the edge resources. Therefore, our algorithm first tries to place each Pod without increasing

the total placeholder size in the system, keeping in mind the delay requirement (Lines 15 and 26 of Algorithm 1). When we can not find any solution that keeps the total backup resource size intact, we have to deploy the Pod first and then create a new placeholder or increase an existing placeholder's size for the Pod (Line 28 of Algorithm 1). Our node selection strategy for Pods favors dispersing them among nodes, leading to a balanced utilization in the system, which in turn decreases the necessary placeholder resources to support single node failures (Lines 20 and 22 of Algorithm 1). In contrast, the placement of placeholders favors those nodes that have a high number of nodes in their vicinity in terms of delay: these "central" nodes are good choices for placeholders, since they can support Pods on many nodes around them. The scheduler does not cover a Pod with a placeholder if the available computational resources do not allow the placeholder creation or size increase, or the delay requirement is so strict that only the starting node appears in the radius (Line 17 of Algorithm 1).

3.2.1 Complextiy analysis

The scheduler algorithm processes the incoming Pod requests at the time of their arrival. Therefore, in a globally available system, the scheduler component need to act fast when a request comes in. We state that our scheduler runs in polynomial time, which we state in Lemma 3.1.

Lemma 3.1. *Our proposed online scheduling algorithm has polynomial complexity.*

Proof. Let us denote the set of nodes with N and the set of Pods with P. In the beginning, getting the nodes in the radius around the origin can be done in $O(|N|)$. Then, the online scheduling algorithm tries to deploy the incoming Pod without increasing the total placeholder size in the system. Regarding that, the algorithm collects the placeholders in the radius and checks the network and computational constraints. This collection and constraint check have the following complexity: $O(|N|^2 + |P|^2)$. In the next step, the algorithm sorts the nodes based on their number of deployed Pods and their number of network connections that fulfill the delay requirements. In the worst case scenario, the algorithm has to do this sorting two times, which means, its complexity can be approximated with $O(2|N| \log |N|) = O(|N| \log |N|)$. After the sorting, the selection of the best fitting node and the deployment takes constant time. To summarize, the complexity of our online Pod scheduling algorithm (without migration) can be approximated by $O(|N| + |N|^2 + |P|^2 + |N| \log |N|) = O(|N|^2 + |P|^2)$, which equals with $O(|N|^2)$ when $|N| > |P|$, and $O(|P|^2)$ when $|N| < |P|$. □

There are certain dynamic operational challenges that scheduling algorithms must face; for remedy we propose a migration policy. Network-aware migration of deployed Pods is triggered when a new Pod request comes in, but the available resources are not sufficient. In these situations we migrate the affected Pods to new nodes to avoid disruptions. The major steps of migration, and the flow of the process between them is presented in the "Migrate" box in Figure 3.2. Although we strive to make room for the incoming Pod in the system, we migrate Pods only if

their relocation frees enough resources and their assigned placeholders' size remains the same.

While the online scheduling will inevitably lead to suboptimal resource allocation for the placeholders, i.e., more resources will be dedicated to backup than the absolute minimal amount at the highest attainable multiplexing scheme for single node failures, we are not sure how often migration events will need to take place. As the authors of [53] argue, edge computing is the strongest candidate for providing low-latency responses, but it is not yet clear what edge infrastructures will be like. In addition to that, the edge applications' dynamics and their latency requirements will greatly affect the frequency of migrations.

Our solution can also handle topology changes dynamically. The fail-over process is triggered, when our scheduler perceives that a worker node is unreachable, or a delay deterioration in the infrastructure spoils Pods' delay constraints. In these cases we use the already provisioned placeholders to restart the respective Pods within their placeholders' resources. After the restart, we remove the Pods from their original placeholders, and try to find or create new placeholders for them.

Both Pod migration and fail-over appear in our online algorithm. Since we consider the delay requirements as hard constraints, both of these methods take the delay requirements into account. In every system, the migration of virtual entities, e.g. Pods, is an expensive process in terms of execution time and operational steps. Although it is a costly operation, we show that our Pod migrating algorithm runs in polynomial time, and we prove its polynomial complexity in Lemma 3.2.

Lemma 3.2. *The migration calculation in our scheduler has polynomial complexity.*

Proof. Let us denote the set of nodes with N and the set of Pods with P. In the migration process the algorithm knows the new Pod (that cannot be deployed in the system due to lack of resources) characteristics and the nodes that are in the latency radius of the Pod's origin node. Our solution iterates over all those nodes' Pods and try to migrate them till one of the nodes has enough free resource to host the new Pod. This means in the worst case we have to try the migration in $O(|P|)$ times. When we examine a Pod if its "migratable", we check the following constraints: i) the actual node will have enough free resource for hosting the new Pod, in case we migrate the examined Pod to another node (can be done in $O(1)$); ii) at least one of the nodes in the examined Pod's radius has enough free resource for that Pod ($O(|N|)$); iii) when a placeholder is assigned to the examined Pod, we do not have to increase its size if we deploy the Pod to a new host (if $|P| > |N|$ then $O(|P|^2)$, otherwise $|N|^2 log|N|$). If all constraints are met, we migrate the examined Pod, so we can deploy the new Pod to its original host. The complexity of Pod migration is $O(|P||N|^2 log|N|)$ in cases, when $|P| < |N|$, else ($|P| > |N|$) it is $O(|P|^3)$. □

As for the technical migration overhead, we argue that stateless [114] application components can be migrated with minimal extra resources. The stateless design, of course, must be supported by a distributed cloud database [113, 115], which transforms the punctual migration overhead into a continuous synchronization of application states onto multiple database instances running on nodes potentially

hosting the stateless application, which leads to an extra consumption in terms of compute, memory and network resources.

3.2.2 Approximation bound

We prove that our scheduler is a 3-approximation algorithm in terms of the amount of placeholder allocation for Pods. As the first step, let us create a graph $G = (V, E)$, where the vertices represent the nodes and the edges of the graph present the connection between the nodes. We denote the set of Pods as P. In the proofs of the approximation we use the graph's diameter $d(G)$, which is the length of $max_{u,v \in V} d(u, v)$ the "longest shortest path" between any two graph vertices (u, v), where $d(u, v)$ is the distance between the vertices. We define the group of vertices that we call *buds* in Definition 3.2.

Definition 3.2 (Bud)**.** A vertex is a bud, if it connects to at least one leaf.

Furthermore, we make an assumption about the graph model and the latency requirements of the Pods in order to render the approximation analysis of our scheduler algorithm analytically tractable. The first part of the assumption is about the size of the topology and the resource capability of each node. The second part simplifies the number and the requirement of Pods to be deployed.

Assumption 3.1. G is a simple, connected graph, with $|V| = n > 3$, each vertex in G represents a node in the Kubernetes [66] cluster and has infinite capacity. Edges in G represent unit latency distance between the vertices. $|P| = |V|$, moreover each Pod $p \in P$ has unit resource requirement, i.e., homogeneous Pod sizes, and there is a one-to-one mapping between the Pods' origins and the vertices in the graph: $p_i \rightarrow v_i; p_i \in P, v_i \in V$, i.e., every vertex is origin for a Pod. The delay requirement of each Pod makes the neighboring nodes of the Pod's origin eligible, no other nodes, i.e., nodes farther than 1 hop yield too much delay for the service deployed in the Pod.

Note that in the following we consider Assumption 3.1 to hold. It is partly a relaxing assumption, e.g., in terms of Pod-, and placeholder placement as infinite node capacities are supposed, but partly specific, e.g., in the aspect of origin selection. In terms of latency requirements, the assumption considers an extremely restrictive scenario.

The goal of an economical scheduler is to find the minimum amount of placeholders that can support all Pods in the system in case of one node's failure. Let us denote by OPT the optimal solution and by $HEUR$ the solution that our online scheduler algorithm yields. Let us denote the number of buds as b, and the diameter of the graph d. The lower bound of the optimal solution can be deduced from the number of buds and the diameter of the graph. Therefore, we define the lowest amount of placeholders that can be theoretically achieved in Lemmas 3.3 and 3.4 using the diameter and number of buds respectively.

Lemma 3.3. $OPT \geq \frac{d+1}{3}$

Proof. G with diameter d has at least one shortest path with length d and must have $d+1$ nodes. Thus, there is a subgraph G' in G that can be represented as a path graph, which has $d+1$ vertices. The Pods' delay requirement allows only the origin node and its neighbors (see Assumption 3.1) as their hosting node. Since every node is an origin for a Pod, the number of vertices in each Pod's radius (whose origin is in G') is 2 or 3 in G'.

In the path graph representation the minimum number of sets that cover all nodes at least once and each set contains only neighboring nodes, equals to dividing the nodes into groups of three. One can see that the number of sets gives the minimum number of placeholders in G, that should be deployed. □

Lemma 3.4. $OPT \geq b$

Proof. We know that a bud is connected with at least one leaf, and each Pod's latency constraint allows only the neighbors of the origin node. Therefore, only two nodes (a bud and the leaf) are in the radius of the Pods, whose origin node is a leaf. Regarding that, one of the nodes in each bud-leaf pairs must hold a placeholder. From this statement, one can see that the number of placeholders must be greater or equal to the number of buds. □

We state, with Lemma 3.5, that our heuristic solution will have at least one Pod, which shares its placeholder with at least one other Pod.

Lemma 3.5. $HEUR \leq n-1$.

Proof. $HEUR \leq n$, as $|P| = n$. By the heuristics applied in our online scheduling algorithm, equality occurs only in the case when placeholders cannot be multiplexed. This would occur only in a G with 1-degree vertices, which is impossible with $n > 3$, hence the statement. □

In order to prove the approximation bound of our scheduler, we have to identify the proportion between: i) the diameter and the optimal amount of placeholders; ii) the number of buds and the amount of placeholders provided by our heuristic solution. Therefore in Lemma 3.6 and Lemma 3.7 we prove that the number of placeholders is directly proportional with the number of buds and the diameter value, as well.

Lemma 3.6. *In case of a fix number of vertices ($|V| = n$) and a fix diameter (d), with the increase of the number of buds (b), the optimal solution (OPT) monotonically increases.*

Proof. Referring on the proof of Lemma 3.4, one can see that the number of placeholders is directly proportional to the number of buds. □

Lemma 3.7. *In case of a fix number of vertices ($|V| = n$) and a fix number of buds (b), with the increasing diameter of the graph d, the number of placeholders, given by our heuristic algorithm ($HEUR$) monotonically increases.*

Proof. Let us induce a path graph G' from a path in G whose length equals to d. We can give a sequence of Pod requests, for which the amount of placeholders provisioned by our heuristic solution would be equal to $HEUR = d$. Since G' has $k = d + 1$ vertices, the $HEUR = d = k - 1$ solution is the worst solution that our algorithm can give on G' (Lemma 3.5). Regarding this, one can see that with the increase of a graph's diameter, the number of placeholders, given by our heuristic algorithm, monotonically increases. □

Simple, connected graphs can have diverse combinations of diameter value and number of buds that affect the number of placeholders provisioned in the system. In Lemma 3.8 we present the possible graph architectures that simple, connected graphs can have with diverse diameter and bud value combinations.

Lemma 3.8. *We deduce the possible number of buds in simple, connected graphs with a given diameter.*

1. *If* $d = 1$, *then* $b = 0$;
2. *If* $d = 2$, *then* $0 \leq b \leq 1$;
3. *If* $3 \leq d \leq \frac{n}{2}$, *then* $0 \leq b \leq \frac{n}{2}$;
4. *If* $d = \frac{n}{2} + k, k > 0 (\frac{n}{2} + 1 \leq d \leq n - 3)$, *then* $0 \leq b \leq (\frac{n}{2}) - k + 1$;
5. *If* $d = n - 2$, *then* $1 \leq b \leq 3$;
6. *If* $d = n - 1$, *then* $b = 2$.

Proof. The indices of the following proofs refer to the indices of cases listed in the lemma.

1. Only complete graphs (from the class of connected, simple graphs) have diameter 1. Complete graphs do not have any leaf vertices, therefore they do not have buds either.

2. The diameter of a star graph, with one central node is 2. This graph has exactly 1 bud node (the central node), since all other nodes are leaves. We can construct several different graphs that have diameter $d = 2$ and 0 buds. One trivial example is a graph that is constructed from a complete graph by deleting a single edge. There cannot be more buds in graphs with $d = 2$, since the smallest combination of vertices and edges where $b = 2$ is a leaf-bud-bud-leaf subgraph that already has $d = 3$.

3. Let us create a cycle with n vertices. This cycle has $d = \frac{n}{2}$ and $b = 0$. We can add extra diagonal edges to this graph so that it will have any diameter value between 3 and $\frac{n}{2}$ and the number of buds remains 0.

   Now let us present the other cases, when the graphs have $0 < b \leq \frac{n}{2}$. For every diameter value $3 \leq d \leq \frac{n}{4}$ we can create a cycle C with $\frac{n}{2}$ vertices and adding extra diagonal edges so that G has $3 \leq d \leq \frac{n}{4}$. At this point, we can

connect the remaining $(G-C)$ $\frac{n}{2}$ vertices (from "outside of the circle") to the cycle so the number of buds is $0 < b \leq \frac{n}{2}$.

We can also construct graphs for which $\frac{n}{4} < d \leq \frac{n}{2}$ and $0 < b \leq \frac{n}{2}$. Let us create a path graph with $d+1$ vertices. The number of internal nodes (that are neither buds, nor leaves in the initial path graph) is $d+1-4 = d-3$. We can add $d-3$ extra leaves connected to the internal nodes in order to increase b. To increase further the value of b, we can connect extra bud-leaf node pairs to any of the internal nodes till we reach the desired number of buds, or we consumed all the vertices in the graph. In case we reach the desired number of buds, but there are more unconnected vertices, we can construct a clique from them and connect it to any internal node.

We showcased possible diameter and bud combinations between the upper and lower bounds of both the diameter and the number of buds. We also showed how we can construct graphs that have any d and b values between the defined bounds. Since the maximum number of buds in a graph with n vertices is $\frac{n}{2}$, there is no other combination that can be constructed regarding the given diameter range.

4. Since the diameter of the graph is greater than $\frac{n}{2}$, the graph can not be a cycle. Let us construct a path graph with $\frac{n}{2}+k+1$ vertices (so the diameter equals to $\frac{n}{2}+k$). Following on Lemma 3.8, we can close the two ends of the path graph with two vertices, and connect the remaining $(\frac{n}{2}-k-3)$ vertices so that the graph will not have any buds. After the induction of the path graph with $\frac{n}{2}+k+1$ vertices, it has $\frac{n}{2}+k+1-4 = \frac{n}{2}+k-3$ vertices and we have $\frac{n}{2}-k-1$ unconnected nodes. The maximum number of buds that is achievable in these scenarios is $\frac{n}{2}-k-1+2 = \frac{n}{2}-k+1$, since the path graph already has 2 buds, one at each end.

5. Let us construct a path graph with $n-1$ vertices with $d = n-2$ and $b = 2$. The remaining one node can be connected to the graph in three possible ways: i) the graph will have $b = 1$ bud, if we form a triangle at one end of the path graph (so we connect the last node to a leaf and to the connected bud); ii) the graph will have $b = 2$ buds, if we connect the last node to two adjacent internal nodes; iii) the graph will have $b = 3$ buds, if we connect the last node to one of the internal nodes.

6. A path graph with n nodes is the only connected, simple graph that has $d = n-1$. This graph must have $b = 2$ buds, i.e., the second vertex on both ends.

□

From Lemma 3.6 and Lemma 3.7 we can draw the following relationship:

$$\max\left(\frac{HEUR}{OPT}\right) \propto \max\left(\frac{d}{b}\right).$$

Based on this observation, we define the approximation bound of our heuristic solution in the combinations of diameter and the number of buds where the latter is minimal and the former is maximal.

Lemma 3.9. *The approximation ratio between our heuristic and the optimal solution for given maximal diameter and minimal number of buds:*

1. *if* $d = 1$ *and* $b = 0$, *then* $HEUR = OPT = 2$;
2. *if* $d = n - 3$ *and* $b = 0$, *then* $HEUR \leq 3OPT$;
3. *if* $d = n - 2$ *and* $b = 1$, *then* $HEUR \leq 3OPT$;
4. *if* $d = n - 1$ *and* $b = 2$, *then* $HEUR \leq 3OPT$.

Proof. The indices of the following proofs refer to the indices of cases listed in the lemma.

1. The optimal solution in every complete graph is 2. The anti-affinity requirement hinders to have only one placeholder. Our heuristic algorithm also achieves the value of 2, since after the deployment of the first request there will be one Pod and one placeholder in the system. Let us denote the hosting node of this placeholder with v. Every other Pod whose origin differs from v will be placed in the system without the need of increasing the number of the total placeholders. For the Pod, whose origin host is v, our solution creates the second placeholder, and deploys the Pod on another node. Therefore, with any order of Pod submissions, the heuristic algorithm will deploy only two placeholders.

2. Let us construct a path graph with $n-2$ nodes with $d = n-3$ and $b = 2$. Now let us close both ends of this path graph with two triangles made by the last two nodes on both ends and the two unconnected nodes. Therefore, the graph will have $b = 0$. In this case the optimal solution equals to $OPT = \frac{d+1}{3} = \frac{n-2}{3}$ (following on Lemma 3.3). Our heuristic solution will deploy maximum two placeholders in each triangle, so at least one of the Pods (per side) whose origin is in the triangle will benefit from a previously deployed placeholder, i.e., two Pods will share a placeholder on both ends of the graph. From this, we can state that at least two nodes will not have any placeholder on them, so $HEUR \leq n - 2$. Therefore, $HEUR \leq 3OPT = n - 2 \leq 3\frac{n-2}{3}$.

3. Regarding to Lemma 3.3, the optimal solution can not be less than $OPT = \frac{d+1}{3} = \frac{n-1}{3}$. Therefore, even when the heuristic solution achieves the worst solution ($HEUR \leq n-1$), we can state that $HEUR \leq 3OPT = n-1 \leq 3\frac{n-1}{3}$. One can see that our statement $HEUR \leq 3OPT$ is proven in graphs with $d = n - 2$ and $b = 1$.

4. The only connected, simple graph that has $d = n - 1$ and $b = 2$ is the path graph with n nodes (see Lemma 3.8). The Pods' delay requirements allow only the origin node and its neighbors (see Assumption 3.1) as their hosting node. Therefore, the number of vertices in each Pod's radius is 2 or 3. Therefore, the

minimum number of sets that cover all nodes, each set containing only nodes that are connected with each other, give the optimal solution for the number of placeholders. Therefore the optimal solution is $OPT = \lceil \frac{n}{3} \rceil$ in a path graph with n vertices. In Lemma 3.5 we showed that the worst case result of our heuristic algorithm is $HEUR = n - 1$ (which is the case in a path graph), hence $HEUR \leq 3OPT = n - 1 \leq 3\frac{n}{3}$.

□

To summarize the previously presented results, we state and prove the approximation bound of our scheduler algorithm in Theorem 3.1.

Theorem 3.1. *Our online scheduling solution is a 3-approximation algorithm for providing joint placement of placeholders of Pods ($HEUR \leq 3OPT$).*

Proof. In Lemma 3.9 we prove that on all possible inputs, the approximation ratio between our heuristic solution and the optimal solution is always less than or equal to 3. Therefore, our scheduler is a 3-approximation algorithm in terms of the amount of placeholder allocation under Assumption 3.1. □

3.3 Re-scheduler: an offline orchestrator to minimize provisioned backup resources

Operating besides the scheduler, our re-scheduler is responsible for the offline minimization of the total provisioned backup resources in the system. The main difference between the two solutions is in the submission pattern of the Pods. While the scheduler works in an online manner, the re-scheduler better approximates the minimum amount of necessary placeholders as it works in an offline manner and it is fed with the batch of all deployed Pods.

Our re-scheduler has three major phases: i) placeholder deployment; ii) Pod deployment; iii) repair phase. The flowchart of the phases are presented inside the "Re-scheduler" box in Figure 3.2. As for the first phase, according to our intuition, the nodes that could host the most Pods are the best choices for placeholders: placeholders on them can cover all those Pods if they are placed elsewhere, which maximizes the multiplexing effect, hence the least possible resources reserved for placeholders. Therefore in the first phase, as shown in Algorithm 2, we reserve the minimum amount of placeholders on the nodes (Lines 2 and 3 of Algorithm 2) that could possibly host all Pods to be deployed.

Algorithm 2: Offline placeholder deployment

```
Data: pods, nodes
Result: pod.placeholder are set for all Pods
1  while pods do
2  |  node ← nodes.sort(by_number_of_deployable_pods, decreasing_order,
   |   pods).first ;
3  |  placeholder ← create_placeholder(node,
   |   maximum_capacity(node.deployable_pods)) ;
4  |  for pod in node.deployable_pods do
5  |  |  patch(pod, placeholder);
6  |  |  pod.placeholder ← node;
7  |  end
8  |  pods.remove(node.deployable_pods);
9  |  node.deployable_pods ← ∅;
10 end
```

The deployment of Pods that can be hosted only on a subset of nodes, e.g., in a strict latency radius, is challenging. The order of Pod deployment follows the number of possible nodes that could host a Pod (Line 6 of Algorithm 3), which mainly corresponds to the tightness of their delay requirements. We deploy the Pods with the fewest options first, then move forward to Pods with looser latency requirements. At the end of this phase, each Pod is deployed and all of them are covered with a placeholder as Algorithm 3 indicates.

Algorithm 3: Offline Pod deployment

```
Data: pods, nodes
Result: pods are deployed
1  foreach pod in pods do
2  |  nodes_in_radius ← nodes.filter(pod.origin, pod.delay);
3  |  suitable_nodes ← nodes_in_radius.filter(pod.capacity);
4  |  pod.choosable_nodes ← suitable_nodes.filter(pod.placeholder);
5  end
6  foreach pod in pods.sort(by_number_of_choosable_nodes,
   |  decreasing_order) do
7  |  chosen_node ← min_utilized(pod.choosable_nodes);
8  |  bind(pod, chosen_node);
9  end
```

Pod migration is an expensive operation since during the migration the behavior of the application can be non-deterministic and the service provider has to guarantee the seamless relocation of the components. Therefore, the cost of migration is not negligible in the minimization process in our re-scheduler. When the re-scheduler is triggered, a deployment that defines the host node, determined by the online scheduler, is in effect for each submitted Pod. Relying on that predefined deployment, in the second phase the scheduler strives to deploy the Pods on those nodes that already host the Pod. Due to this behavior, our solution can minimize the number of migrations while still minimizing the amount of total provisioned resources.

It is possible that some of the reserved placeholders' size might not be enough to back up all the Pods which have been assigned to them. In order to ensure full reliability, we have to repair those failed placeholders, one input to Algorithm 4. Since we minimize the total footprint of provisioned placeholders, we reassign Pods from each failed placeholder to other placeholders, or re-deploy them to other nodes if migration is favored (Line 2 of Algorithm 4). If we can not find any solution that would keep the amount of total placeholder size on the same level, we have to increase the broken placeholders' size (Line 8 of Algorithm 4), or instantiate new placeholders (Line 14 of Algorithm 4).

Algorithm 4: Offline repair

```
   Data: bad_placeholders, nodes
   Result: no bad placeholders left
 1 foreach placeholder in bad_placeholders do
 2 |   success ← reassign(placeholder, nodes) ;
 3 |   if success then
 4 |   |   bad_placeholders.remove(placeholder);
 5 |   end
 6 end
 7 foreach placeholder in bad_placeholders do
 8 |   success ← increase(placeholder) ;
 9 |   if success then
10 |   |   bad_placeholders.remove(placeholder);
11 |   end
12 end
13 foreach placeholder in bad_placeholders do
14 |   success ← create_placeholder(placeholder) ;
15 |   if success then
16 |   |   bad_placeholders.remove(placeholder);
17 |   end
18 end
```

Although our re-scheduler works in an offline manner, i.e., it processes the batch of all the deployed Pods' requirements, we must not let its execution time increase unpredictably. The state of the system is continuously changing: Pods come and go, nodes might fail. If such events occur while the re-scheduler is running, the placement result yield by the algorithm may not be valid anymore. Therefore it is of paramount importance to design the re-scheduling algorithm to be fast. In Lemma 3.10 we prove that our proposed re-scheduler algorithm runs in polynomial time.

Lemma 3.10. *Our proposed re-scheduler algorithm has polynomial complexity.*

Proof. Let us denote the set of nodes with N and the set of Pods with P. In the re-scheduler algorithm's first phase we deploy the placeholders. As the first step, it calculates the nodes in each Pod's radius. This calculation can be estimated with $O(|P||N|)$. After the calculation, the re-scheduler sorts the nodes by the number of the Pods that could be hosted on them. The complexity of this sorting is

$O(|N|^2)$. In the next step, it deploys each placeholder and reorders the list after every deployment. This step and the whole first phase can be estimated with $O(|P||P||N|(|N|-1)) = O(|P|^2|N|^2 - |N|) = O(|P|^2|N|^2)$.

In the second phase the re-scheduler places the Pods on the nodes. First, it sorts the Pods by their number of fitting nodes. The complexity of the sorting is $O(|P|^2)$. Then, the algorithm iterates through the sorted Pods, and deploys them on the least utilized node. The worst case complexity of this iteration can be estimated by $O(|P||N|)$. The worst case complexity of the whole second phase, if $|P| > |N|$, is $O(|P|^2)$, anyway the worst case complexity is $O(|P||N|)$.

In the third phase, the algorithm checks if any broken placeholders exist, and repairs the failed ones. When it checks the placeholder constraints, it iterates through all the placeholders and for each of them it also iterates through all the nodes and the Pods. The worst case complexity of this validation is $O(|P||N|^2)$, as in the worst case each node contains one placeholder. Then, the re-scheduler goes through all the broken placeholders, and tries to fix them with Pod reassignment to different, already instantiated placeholders, or to other nodes. During the reassignment for each broken placeholder, the algorithm gets those nodes whose constraints are not fulfilled and tries to reassign the specified Pods. In case of $|P| > |N|$, the reassignment's complexity is $O(|P||N|^2)$, otherwise the complexity is $O(|N|^3)$. In the second repair method the algorithm increases each broken placeholder's size, if possible, which has the worst case complexity of $O(|P||N|)$. The last repair attempt is the new placeholder, which has the same worst case complexity as the Pod reassignment $O(|P||N|^2)$or $O(|N|^3)$. If $|P| > |N|$ the worst case complexity of the third phase equals to $O(|P||N|^2 + |P||N|^2 + |P||N| + |P||N|^2) = O(|P||N|^2)$, otherwise it is $O(|P||N|^2 + |N|^3 + |P||N| + |N|^3) = O(|N|^3)$.

Summarizing, our proposed heuristic algorithm's complexity in the worst case scenario equals $O(|P|^2|N|^2 + |P|^2 + |P||N|^2) = O(|P|^2|N|^2)$ in case of $|P| > |N|$, otherwise $O(|P|^2|N|^2 + |P||N| + |N|^3) = O(|P|^2|N|^2 + |N|^3)$. □

3.4 Providing scalability with node clustering

As the size of the system grows, not only finding the best placement for the service components, but even measuring the network characteristics becomes challenging. The continuous measurement is induced by the fact that the underlying network and topology may change, and such events can cause application failures and delay requirement violations. Our clustering solution not only reduces the overhead of determining network characteristics, but also helps the scheduling of the applications by compressing the topology. By topology compression, we mean that since a clustering algorithm forms groups (clusters) from a set of nodes, the scheduling algorithms do not need to iterate through all the nodes, it is sufficient to calculate with the clusters. Furthermore, our solution provides valuable input for service providers who want to implement self-organizing network features to dynamically organize their federated system hierarchically, based on their network topology.

We propose a clustering algorithm that groups the physical nodes into clusters in order to ensure that dynamic application placement and network delay measurements

can scale effectively in large topologies. Our solution is an agglomerative clustering algorithm that creates cluster layers hierarchically, where each layer contains clusters that are constructed with a new delay requirement belonging to a Pod request. The input of our clustering algorithm is a topology (that maybe already clustered before), and a set of delay values that will be used for clustering the nodes (or the clusters) inside topology. In this agglomerative clustering approach, each node starts in its own cluster, then the clusters are merged as we build up the hierarchy, where each layer (and their clusters) are built based on increasing delay requirement values.

In cases when the topology is not clustered with a given delay value yet, a new layer is being created (visualized in Figure 3.2, in "New layer creation" box inside the "Clustering" box) relying on the underlying layer that is clustered with the delay that is the greatest and closest, but still less than the new one. Regarding the application placement, our clustering mechanism guarantees that all nodes inside a cluster fulfill a Pod's delay requirement in case the cluster is an output of the clustering with that delay value. Hence service delay requirement violations can not happen inside a cluster for the given delay value, no matter which member node hosts the given Pod.

The clustering may lead to different outcomes, in which a node may belong to different clusters. An illustrative example can be viewed in Figure 3.3, where on the top, a simple topology is depicted with delays on network connections between the nodes. On the bottom, we present how the topology can be clustered based on different delay requirements. The red circled clusters (bottom middle) are non-deterministic, since they overlap each other and the clustering result depends on the processing order of the nodes.

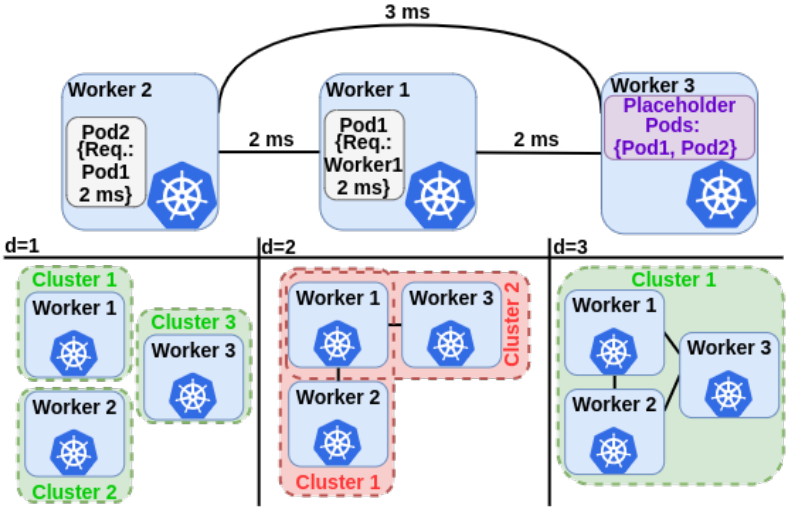


Figure 3.3: Deterministic and non deterministic clustering examples

There are some delay values (denoted as d in Figure 3.3), for which the clustering is deterministic, i.e., each node belongs to only one determined cluster no matter the order of nodes during the clustering process. Formally we define this problem in Problem 3.1.

Problem 3.1 (Deterministic clustering problem)**.** Given a $G = (V, E)$ graph and a d delay value. G is an undirected complete graph with weighted edges that fulfill the triangle inequality and d is a positive number that represents the delay requirement of a service.

Can G be clustered based on d in a deterministic manner?

A key feature of our solution is to find those delay values (for the given topology) that provide such deterministic clustering results. We call these delays as *vantage-free delays* and we seek these delays at the initialization of the clustering component ("Initialization" box inside "Clustering" box in Figure 3.2). These *vantage-free* clustering layers can be defined and created in polynomial time by using only the delay values that appear between the nodes in the topology (see Lemma 3.11).

Lemma 3.11. *We can find an answer for Problem 3.1 in polynomial time.*

Proof. Let us represent our topology with a complete graph G, where the nodes are the vertices and the edges are the smallest available latency values between each node pair. The proof consists two steps. First, we delete the edges from G if their weight is greater than d, in $O(|V|^2)$. Then, we examine each disconnected component, if they are complete subgraphs. This examination can be done in $O(|V|^2)$. If all of the disconnected components are complete subgraphs (cliques), the output is positive (yes), G can be clustered based on d deterministically, otherwise it cannot. □

The purpose of defining these delay values and their clustering layers is that they serve well as underlying layers for clustering with other, non *vantage-free delay* values, since these *vantage-free delays* do not change unless the topology changes. In order to support the large scale scheduling, the purpose of our clustering solution is to create the least clusters that cover all the nodes and the nodes cannot violate the given delay requirement within their cluster. We define this problem in Problem 3.2.

Problem 3.2 (Delay based clustering problem)**.** Given a graph $G = (V, E)$ that represents the physical topology, and d, a positive number that equals to the delay requirement of the service. G is an undirected complete graph, whose edges fulfill the triangle inequality.

Clustering the vertices with minimum number of clusters with the awareness of the following requirements: i) a cluster has to be a clique; ii) the weight of every edge inside the clusters is less than or equal to d; iii) clusters can not overlap with each other.

Theorem 3.2. *Problem 3.2 is NP-hard.*

Proof. In the proof of this theorem, we use Karp-reduction for a known NP-hard problem, the Clique cover problem, which is the algorithmic problem of finding a minimum clique cover. A clique cover of a given undirected graph is a partition of the vertices into cliques.

As a preparation step, we construct G' with deleting all of the edges with greater weight than d from G, since those edges surely will not appear inside any cluster. After this step, we can ignore the edge weights in G'. In this case we strive to

find the minimum number of not overlapping clusters that are cliques, and cover all the vertices. Note that, since service delay requirement violations can not happen inside a cluster for the given delay value, the clusters can only contain cliques in our solution. Let G' be the input of the clique cover problem. In this case, one can see that finding the minimum clique cover (the clique cover that uses as few cliques as possible), equals with finding the minimum clusters (that are also cliques) that solves Problem 3.2. Also a solution for Problem 3.2 gives a good clique cover for the MINIMUM CLIQUE COVER PROBLEM. Since G' can be constructed in polynomial time ($O(|V|^2)$) from G, and covering with minimum clusters on G is fully complaint with CLIQUE COVER PROBLEM on G', Problem 3.2 is NP-hard. □

We proved that Problem 3.2 is a hard problem in general, although in some cases it is solvable in polynomial time. In Theorem 3.3 we state that Problem 3.2 is solvable in polynomial time in cases when Problem 3.1 gives positive answer for the same input.

Theorem 3.3. *On a given complete, weighted graph G (whose edges fulfill the triangle inequality) and d positive number for which answering Problem 3.1 gives positive outcome (yes), a solution can be found for Problem 3.2 in polynomial time.*

Proof. In Lemma 3.11 we proved that we can find an answer for the question of Problem 3.1 in polynomial time. In cases, when Problem 3.1 has a positive answer if we delete all edges with greater weight than d from G, then all the disconnected components of G (after the deletion) are complete subgraphs. Since none of the remaining edges has greater weight than d, the optimal solution for Problem 3.2 equals with the disconnected complete subgraphs. □

When our clustering component is initialized, the scheduler receives the node clusters from our clustering component when they are looking for nodes in a certain delay radius. The scheduler calls the clustering component with the origin node and the delay radius required by the Pod. If there is already a constructed cluster layer with that delay value, the algorithm finds the appropriate cluster (that holds the starting node), and returns that to the scheduler. If the topology is not clustered with the given delay value yet, the algorithm creates a new layer that is based on the underlying layer that is clustered with the delay that is the greatest and closest to the new one, but still less than the given one. Relying on that underlying layer and its delay matrix, the algorithm creates the new layer in six steps:

1. delete the edges that are greater than the delay requirement;
2. find a maximal clique in the graph;
3. create cluster from the found maximal clique;
4. remove the vertices in the clique from the graph;
5. return to step 2, until all of the vertices are deleted from the graph;
6. create the new delay matrix.

After we defined the clusters with the new delay requirement, a new delay matrix must be created that holds the delay values between the clusters. We apply the conservative complete-linkage clustering method (also known as farthest neighbor clustering) for delay matrix creation to calculate the delay values between clusters. The complete-linkage clustering method means that the delay value between two clusters equals to the delay between those two nodes (one in each cluster) that are farthest away (have the highest delay) from each other.

3.5 Machine Learning-based auto-scaling

We dissect the problem of cloud auto-scaling into three phases. First, given the history of client requests' time series and multiple ML-based forecast methods applicable on them, we design a Kubernetes plug-in that implements the methods and selects the most reliable forecast at all times, based on the evaluation of their accuracy on recent measurements. Second, assuming a potential, but not completely certain request rate that the selected forecast method yields, we adapt the application scale accordingly, in order to meet the application provider's intention on minimizing SLA violations. Third, we experimentally determine the required scale of the specific application given any request rate. This last step is in fact performed prior to the deployment by emulating arbitrary request rates. Here we delve into the third step, and we present the others in later sections.

Measuring the compute resource requirements of applications, i.e., resource profiling, has been investigated by researchers with resource optimization in mind. The authors of [133] for instance built application profiles in order to develop an application-execution policy that minimized the energy consumption of the mobile device. According to their approach, the applications in focus could be executed on the mobile device or in the cloud, and they built profiles for mobile applications based on the size of their input data, and the compilation deadline. In contrast to that work, our goal is to model the resource requirement of applications when serving any given rate of application requests, as accurately as possible. To this end we define application profiles as follows, projected to a Pod, i.e., the smallest runtime and scaling unit in the Kubernetes system. In a Pod we can include a container or a set of containers we want to run.

Definition 3.3 (Application profile)**.** The application profile is an AP:$\mathbb{N}^+ \rightarrow \mathbb{R}^+$ function, which assigns the service rate per Pod to the number of active Pods.

We designed a measurement method to determine the profile for applications hosted in Kubernetes; the components of the measurement system are shown in Figure 3.4. In our Kubernetes cluster we replaced the standard load balancer functionality of the Kubernetes Service object with an ingress controller. We deployed a fortio [37] benchmark tool to a worker node, and the application we wanted to test was deployed to another worker node. During the measurement we run benchmark tests with an increasing number of requests per second. We started the benchmark with one Pod. When a test finished with the actual number of Pods, we increased the Pod count and started the measurement again. Each test took one minute. After

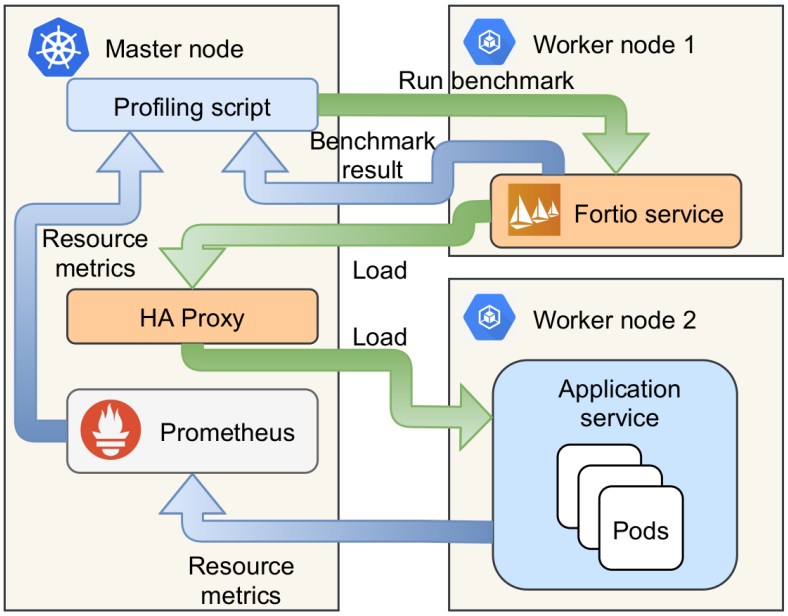


Figure 3.4: Measurement setup for application profiling

each test, we collected the percentage of lost requests from fortio and the average CPU usage from Prometheus, a monitoring tool for Kubernetes [96].

We present results for 3 applications in Figure 3.5 by showing the served query per second (QPS) in function of the number of running Pods. Using the empirical measurement results we can approximate the resource profile of each application. On the top diagram the empirical result of a Node.js application and an Nginx web server can be seen. In the bottom figure the profile of an image classification application is shown. Let us take the measurement results of the Node.js web application that serves static HTML files, depicted in Figure 3.5 with the green dashed line. The empirical results can be approximated with the linear function (depicted with the green dotted line) of $125x + 209$, where x is the number of running Pods. We calculated the R^2 value of this function and we got 0.997. R^2 is a statistical measure that represents the proportion of variance in the outcome variable that is explained by the predictor variables in the sample. Its value falls between 0 and 1; a value close to 1 means that our predictor fully explains the variance in our samples, i.e. the closer we are to 1, the better our prediction [81].

According to the fitted approximation function, the more active Pods are in the system, the lower service rate per Pod the application can provide. From this linear function the profile of the Node.js web application using Definition 3.3 is $AP_{Node.js}(x) = 125 + \frac{209}{x}$. Measuring profiles for a wider set of applications is out of the scope of this paper, but we note that there surely exist applications for which profiles are more complex than a linear relation, e.g., the measurements of the image classification application in the bottom part of Figure 3.5 can be approximated by a quadratic function. We argue though that Kubernetes is predominantly used for web applications, therefore selecting a popular web server like Node.js for a representative profile seems evident. In the following we will use this function as the application profile in our simulations.

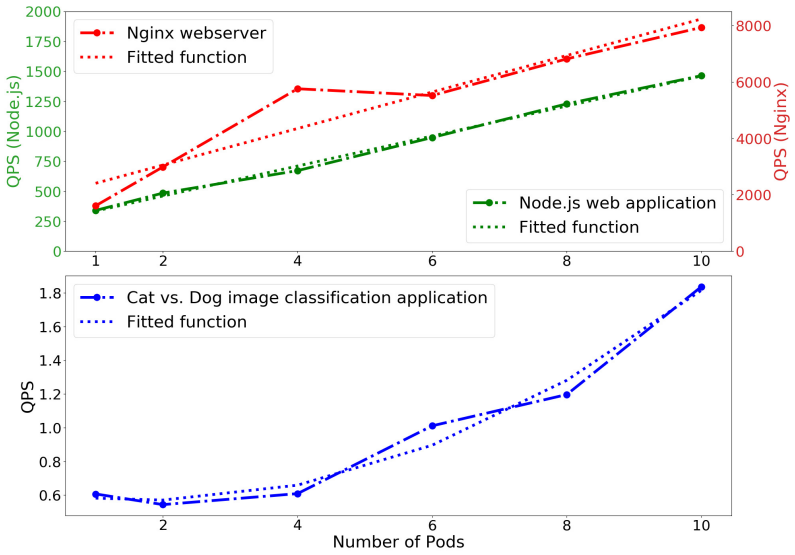


Figure 3.5: Profiles of different applications

3.5.1 Analytical models of auto-scaling methods

In this section we propose analytical models to describe the behavior of Kubernetes' Horizontal Pod Autoscaler (HPA) [52]. The models allow us to run extensive simulations for evaluating our ML-based forecast methods and their scaling accuracy against the baseline HPA.

In Kubernetes, HPA is influenced by several parameters. There are cluster level settings, e.g., down-scaling stabilization, Pod synchronization period and scaling tolerance, and there are HPA level parameters, such as minimum and maximum Pod number, scaling threshold according to arbitrary metrics. By default HPA is based on CPU utilization measurements: HPA periodically fetches monitoring data from the system, and takes a decision on how many Pods the cluster should have. This period is called a scaling interval. The scaling operation and the decision-making procedure is formally described in Definition 3.4.

Definition 3.4 (Kubernetes HPA parameters and operation)**.** The following parameters must be configured for HPA [52]:

1. c_{min} and c_{max} are the minimal and maximal allowed Pod counts,
2. $\hat{u} \in [0, 1]$ is the targeted average CPU utilization over all running Pods,
3. scaling interval is a $\mathbb{R}^+$-long time window, at the end of which HPA evaluates metrics for scaling decisions,
4. $s \in [0, 1]$ is the scaling tolerance.
5. d is the downscale stabilization, i.e., the duration the HPA has to wait before another downscale operation can be performed after the current one has completed.

Let d_i be an indicator function of whether downscaling is enabled or not during the scaling interval $i \in \mathbb{N}$.

$$d_i = \begin{cases} 1 & \text{if stabilization is active (downscale is disabled);} \\ 0 & \text{else.} \end{cases} \tag{3.1}$$

$u_i \in [0, 1]$ is the measured CPU usage in scaling interval $i \in \mathbb{N}$. A scaling decision is made in scaling interval i, if

$$\left| \frac{u_i}{\hat{u}} - 1 \right| > s, \tag{3.2}$$

and the number of Pods in interval $i + 1$ is recursively yield:

$$c^*_{i+1} = \left\lceil c_i \frac{u_i}{\hat{u}} \right\rceil. \tag{3.3}$$

After applying the limits, Pods are terminated or instantiated to meet the count

$$c'_{i+1} = \begin{cases} c^*_{i+1} & \text{if } c_{min} \leq c^*_{i+1} \leq c_{max} \\ c_{min} & \text{if } c^*_{i+1} < c_{min} \\ c_{max} & \text{if } c^*_{i+1} > c_{max}. \end{cases} \tag{3.4}$$

HPA will not perform downscale operation if there was another one in the previous time window of length d.

$$c_{i+1} = \begin{cases} c_i & \text{if } c'_{i+1} < c_i \text{ and } d_{i+1} = 1; \\ c'_{i+1} & \text{else.} \end{cases} \tag{3.5}$$

The higher the d value, the more resources the application uses, because it scales out without such stabilization, but scaling in is slower compared to applying a low d.

For modeling purposes we make some simplification to HPA's operation and to the application profile. We assume that d is equal to the scaling interval, i.e. $d_i = 0 \ \forall i$, and $c_{min} = 1$, $c_{max} = \infty$, $s = 0$. The application profile is assumed to be constant, i.e., $AP(x) = \mu$, μ being the rate of the exponentially distributed service times. We further suppose that the arrival process of requests can be described as a Markov-Modulated Poisson Process (MMPP) with m states; we denote the respective transition matrix by Q and the m arrival rates by $\lambda_1, \lambda_2 \ldots, \lambda_m$.

HPA sets the number of servers periodically, so in our model we make the case for a changing number of servers that is adapted dynamically to the change of the arrival rate. HPA is reactive, i.e., the change of arrival rate takes effect on the number of servers in the next scaling interval. Therefore the number of servers follows the MMPP arrivals, but lags one scaling period. Let $\lambda(t)$ be the arrival rate at time t. By assuming that the time MMPP spends in a particular state (e.g., minutes) is orders of magnitude longer than the time between request arrivals (e.g., milliseconds), the

number of servers at t can be calculated with the following formula:

$$C(t) = \left\lceil \frac{\lambda(t-1)}{\rho\mu} \right\rceil, \tag{3.6}$$

where ρ is the server utilization used in M/M/c queue models, and can be calculated as $\rho = \lambda/(c\mu)$ with λ arrival rate, c servers and μ service rate. The utilization equals to the management parameter of HPA, i.e., $\rho = \hat{u}$.

From this model we can calculate Pod usage to describe the behavior of Kubernetes HPA, and we can analyze the MMPP parameters. The steady-state probabilities π of the underlying Markov chain of MMPP arrivals are given by:

$$\pi = \pi Q \text{ and } \pi\mathbb{I} = 1, \tag{3.7}$$

where $\mathbb{I}$ is an all-ones vector. The steady-state arrival rate is:

$$\lambda_{MMPP} = \pi\lambda^T \tag{3.8}$$

where λ^T is the transpose of the row vector $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$.

Although using the steady-state distribution of MMPP we can infer the arrival rates, and from that the probability of having a given number of Pods in the system, the disadvantage of this model is that it does not account for job losses. As timed-out requests are paramount to count during the operation, we also propose a more complex, discrete-time queuing model for mimicking the calculation of c^*_{i+1} in HPA in subsequent simulations. Our model is similar to the one presented in [59], where the scaling decision was based on the ratio of customers waiting and on the number of servers. Although instead of the number of queued requests, we use the number of served requests in the scaling decision, because the number of served requests and the CPU usage are tightly coupled, and the latter is the basis of scaling decisions of HPA.

Let the number of requests in the system in period i be

$$L_i = L_{i-1} - M_{i-1} + \Lambda_i - T_i \tag{3.9}$$

where Λ_i is the number of arrived requests, M_i is the number of served requests in period i and T_i is the number of queued requests that expire in period i. M_i is defined as follows.

$$M_i = \min\{c_i AP(c_i), L_i\} \tag{3.10}$$

where c_i is the number of Pods and $AP(c_i)$ is the service rate described by the Application profile defined in Definition 3.3, i.e., M_i is the minimum of the number of requests in the system and of the number of requests the system could serve in period i. L_i contains the requests arrived during time period i, but does not contain the requests served in period $i-1$, nor the requests lost in period i. We assume an empty queue in the initial period, i.e., $i = 0$, $L_0 = 0$, $c_0 = 1$, $\Lambda_0 = 0$.

This model expresses request loss with the number of timed-out requests T_i, reflecting under-provisioning of the system in the particular scaling interval. The formula for T_i is given in (3.11), where t denotes the value of timeout expressed in scaling

intervals, e.g., if $t = x$, then all requests arrived in period $(i - x)$ and not served from period $(i - x)$ until period $(i - 1)$ are timed out in period i.

$$T_i = \begin{cases} 0, \text{ if } (i - t) < 0 \\ (L_{i-t} - \sum_{j=1}^{t} M_{i-j})^+ \end{cases} \tag{3.11}$$

CPU usage can be approximated by the ratio of number of served requests over the maximum number of requests that can be served. Therefore the current CPU usage can be written in the following form:

$$u_i \simeq \frac{M_i}{AP(c_i)c_i}, \tag{3.12}$$

which yields the following for (3.3), the number of required Pods in Kubernetes:

$$c^*_{i+1} = \left\lceil \frac{M_i}{\hat{u}AP(c_i)} \right\rceil. \tag{3.13}$$

We introduce downscale stabilization as a positive integer d: if $d = 1$, it has no impact on the system, because scaling operation can not be performed within one scaling interval. The indicator function of downscale stabilization in scaling interval i can be formed as follows for $d > 1$:

$$d_i = \begin{cases} 1 & \text{if } \exists j : 2 \leq j \leq d, c_{i-j+1} < c_{i-j} \\ 0 & \text{else.} \end{cases} \tag{3.14}$$

With this model we can simulate the number of Pods in the system, their resource usage and the number of lost requests, taking into account the unique profile of the application.

There are widely used data sets for testing web applications, the two most common are NASA-HTTP [85] (two months' worth of all HTTP requests to the NASA Kennedy Space Center web server in Florida) and WorldCup98 [134] (three months' HTTP requests made to the 1998 World Cup Web site). We find that these logs are rather outdated (1995 and 1998). Therefore to validate the presented HPA models, we ran simulations with recent anonymized web traffic traces collected in a university campus network for a week in the middle of a fall semester. The dataset contains the flow count per second to Facebook; the trace is shown in Figure 3.6. Since the Facebook data is generated by real users, it is a good example of how the traffic of a cloud application manifests, e.g., it shows daily trends, and the average number of requests per minute is not extremely high, but it shows relatively high variance, hence it is a good fit for the edge cloud application flavor, i.e., relatively small user base, hectic traffic. Therefore we use these traces for our simulations.

We also generated an MMPP-based traffic trace for simulation purposes. An MMPP has been fitted to the Facebook traffic trace using the algorithm described in [99]. The algorithm is able to calculate the parameters of an MMPP from the traffic information. For simulations we used a generated arrival sequence from this fitted

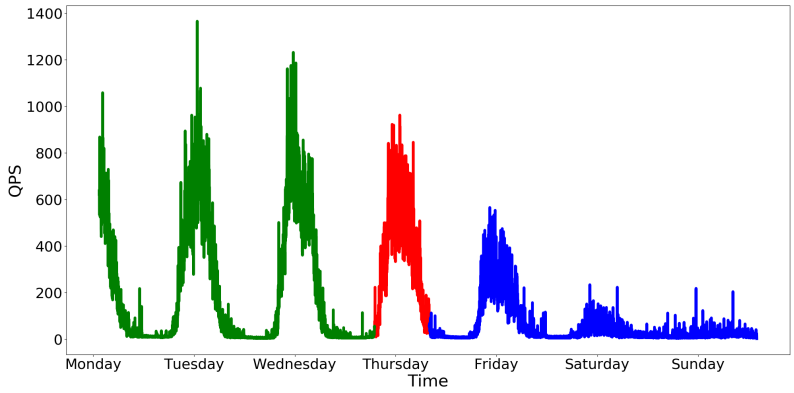


Figure 3.6: One week Facebook traffic in a university campus (green: training data, red: test data, blue: weekend, not used)

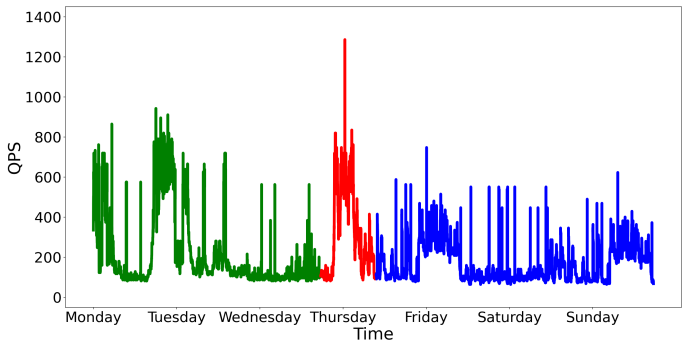


Figure 3.7: One week of MMPP-based traffic generated from the Facebook load using the algorithm described in [99] (green: training data, red: test data, blue: weekend, not used)

MMPP; the final MMPP-based traffic can be seen in Figure 3.7. With this synthetic trace, we are able to compare the MMPP/M/c HPA model with the lossy discrete model. Moreover, it is apparent that the synthetic trace does not show the daily pattern as clearly as the real trace, therefore our forecast methods' (in Sec. 3.5.2) behavior can be simulated and evaluated in case of a less predictable traffic, with which we can analyze the robustness of our system.

First we checked whether the discrete model matches HPA's behavior by comparing the number of Pods started in response to the real trace in a simulator of the model, and in a real Kubernetes cluster. With our model we could simulate the number of Pods in the system, i.e., resource usage, and the number of expired requests, taking into account the unique profile of the application. The scaling interval was set to one minute making the start-up time of new Pods negligible in comparison. The downscale stabilization was set to one minute, too. Results are shown in Figure 3.8: x-axis shows the time, the y-axis of the first diagram depicts the number of Pods, that of the second diagram shows the sum of Pod minutes, i.e., cumulative sum of Pod numbers accumulated through the simulated minutes. As the results suggest, the discrete model behaved similarly to the real HPA operation, i.e., it gave a good approximation of the number of Pods. We calculated the Mean Squared Error

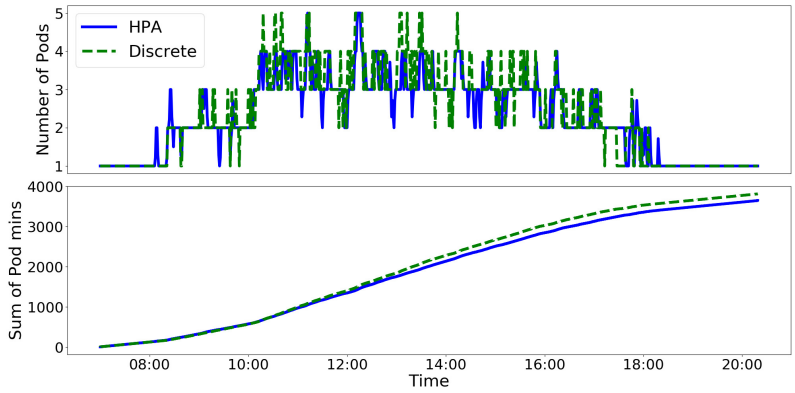


Figure 3.8: Comparison of the lossy discrete model and HPA - Pod usage

(MSE) on the per minute Pod number of the simulated model compared to the measurement, and we got 0.27 while the average number of Pods is 2.38 by the discrete model and 2.28 by the HPA measurement.

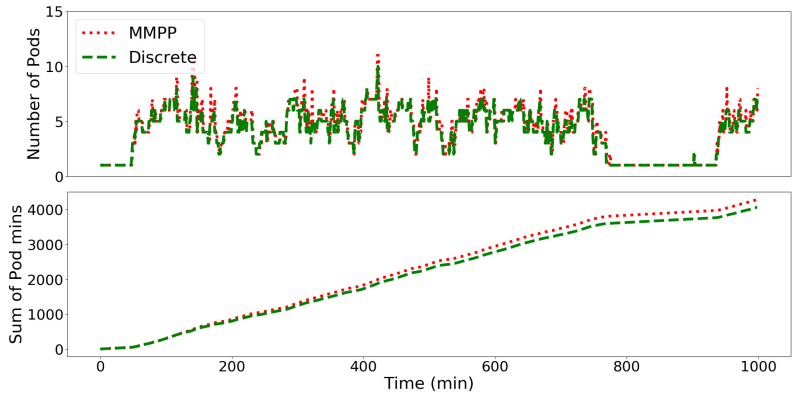


Figure 3.9: Comparison of lossy discrete and lossless MMPP/M/c models - Pod usage

Next we compared the discrete and the MMPP/M/c-based models. Since the latter requires MMPP input, we used our synthetic traffic (Figure 3.7). We applied the following equation to calculate the number of Pods for the MMPP/M/c-based model in scaling interval i: $c_{i+1} = C(i+1)$, where C is the method of calculating Pods given in (3.6). We run the simulations with a constant service rate and a downscale stabilization of less than one scaling interval. The results are shown in Figure 3.9. Like in Figure 3.8, the top chart depicts the number of Pods, the bottom chart shows the cumulative sum of Pod minutes in function of time. The two analytical models behave similarly. Slight difference can be observed when the input traffic is volatile. To quantify the difference, we calculated the average number of Pods and the MSE from the results, and we listed them in Table 3.1. Based on the results for various average utilization values, it can be stated that the MSE is low compared to the average number of Pods.

It follows that, under certain conditions, the MMPP/M/c model can be used instead of the discrete model, enabling the analytical evaluation of the behavior of HPA.

Table 3.1: Comparison of lossy discrete and lossless MMPP models

Pod usage	Utilization ratio		
	0.85	0.9	0.95
Average - lossy discrete	7.18	5.19	4.07
Average - lossless MMPP/M/c	7.31	5.33	4.30
MSE	0.16	0.19	0.29

Since the MMPP/M/c model assumes lossless operation, intuitively the approximation of HPA is better in settings that result in fewer lost requests, which is reached at lower utilization, hence the relatively low MSE value at 0.85.

3.5.2 The proposed proactive scaling engine

In order to achieve more effective resource usage and higher service quality than the reactive HPA, we decided to create a scaling method that anticipates future requests and allocates or releases resources in advance. In our so-called HPA+ solution incoming request prediction and the resulting scaling decisions are implemented in two separate modules. In contrast to many solutions that propose using only Q-learning for server scaling [100, 29, 16], we use popular ML methods for prediction [22, 55] in addition to the Q-learning method: auto-regression and Neural Network (NN). We find it important to utilize models that have substantial differences in their operation. AR is a simple model with low resource requirement, whereas LSTM and HTM require more computing power. However, the latter two models differ in nature: the former requiring supervised learning and the latter being an unsupervised learning approach. Furthermore, the RL method (Q-learning) requires a relatively long time for exploring the state space until good actions (or predictions) are issued.

Our ML models were trained and tested on the anonymized web traffic traces shown in Figure 3.6. We chose to use the time series of requests targeting Facebook, because the traces show typical usage patterns: it is a highly visited website, the number of visits from the campus shows a daily and weekly profile, and the standard deviation of visits per minute is high, which describes well the request dynamics of an average web service with a moderate customer base. To build and evaluate forecast models we standardized the dataset using Z-score normalization. We chose the time granularity to be in the order of minutes because both the web request time-outs and the Pod startup times fall in the order of seconds. Furthermore, using a grid search on the granularity value and other system parameters, we found that the trends in traffic can be best captured using one minute granularity. This observation is also supported by Rattihalli et al. [101] in their work.

Our first model to predict the traffic load was an auto-regressive (AR) model. It assumed that the traffic load at a given time linearly depended on the previous values of the time series. After the training phase we realized that the previous 32-minute observations had to be used for achieving the best accuracy. The coefficients to what extent we use each previous observation was calculated by optimizing the model.

The second model for the time series analysis was chosen from supervised deep learning methods: we used the popular Long Short-Term Memory (LSTM) NN. We examined several values for the size of the look-back window and the most beneficial turned out to be 15-minute long. Thus for LSTM we used a 15-minute look-back window, i.e., all the load values from $t-15$ to t were used to predict the load in $t+1$. We also used the load difference between loads in the look-back window and the number of elapsed minutes from midnight in each t as input to the network.

We picked our third model from the family of unsupervised deep learning: we selected the widely used Hierarchical Temporal Memory (HTM). It is a biological NN, designed to learn temporal patterns from sequences [47]. We used the implementation of Numenta.org in which we encoded the input load and the timestamp so that the network would take into account 15-minute history. Numerous parameters of the HTM architecture were optimized to achieve the best performance.

We used SARSA and Q-learning as our fourth, RL-based approach. In this case, the state of the system is modeled with the current load and the number of running Pods. The RL approach looks for the best action to execute in the given state. The possible actions in a state are: scale out, scale in, idle. Even though the output of the RL solution is the number of Pods required, we can convert this to a predicted load by calculating the QPS served by the given number of Pods.

The training set for the four models consisted of the first 3 days of the week and the test set was the fourth day, as shown in Figure 3.6. Each model was trained on the training set. The hyperparameters of the models were optimized using grid search and walk-forward cross validation, which means that each model was re-trained several times on increasingly larger sets. Then with the best performing hyperparameters the final models were created and then evaluated on the test set. For an illustrative example, we wanted our forecast models to learn the daily profile of workdays; weekends (and Friday afternoon) are significantly different, but the four-day window proved to be sufficiently long for training and testing.

The results of the forecast methods were examined with the root MSE (RMSE) of their prediction and with their R^2 value. The R^2 value assesses how well a model explains and predicts future outcomes. The closer the value is to 1, the stronger the predictive power. The RMSE of the AR, LSTM, HTM and RL were 0.092, 0.107, 0.138 and 0.737 and the R^2 value were 0.879, 0.859, 0.819 and 0.456. Comparing the time needed for training, the AR model was significantly faster than the others, its training time was $14.7ms \pm 0.6ms$. Besides the advantage of being taught quickly, computing the AR model is not resource intensive, but it is sensitive to outliers. On the other hand, LSTM handles outliers well, but its training is more resource intensive ($283s \pm 12s$). In terms of resource requirements, the HTM falls between AR and LSTM ($13s \pm 2s$), however the main advantage of HTM is that it is robust to noise and it is able to learn continuously from each new input pattern, no prior training is required. The important features of RL-based approaches are learning without prior knowledge, and the ability to learn online and update environmental knowledge by actual observations. However the learning time of our RL model is similar to that of LSTM, i.e $212s \pm 24s$.

We further compared the instantaneous performance of our lossy discrete-time queuing model of HPA and that of our ML forecasting models in numerical simulations. We compared them to the oracle: perfectly predicting the number of arriving requests in every scaling interval, and setting the number of Pods accordingly. The oracle is used as a benchmark for resource usage, i.e., we evaluated each model through the excess of their Pod usage, compared to that of the oracle. Furthermore, on the other side of the provisioning decision problem, we looked at the request loss ratio, denoted by Loss in the figures, which shows the ratio of lost requests to the total number of web requests. Again, our four ML-based models (AR, LSTM, HTM, RL) were trained on the 3-day series of flow counts towards facebook.com in our traffic traces, and then the models were evaluated on the fourth day's traffic.

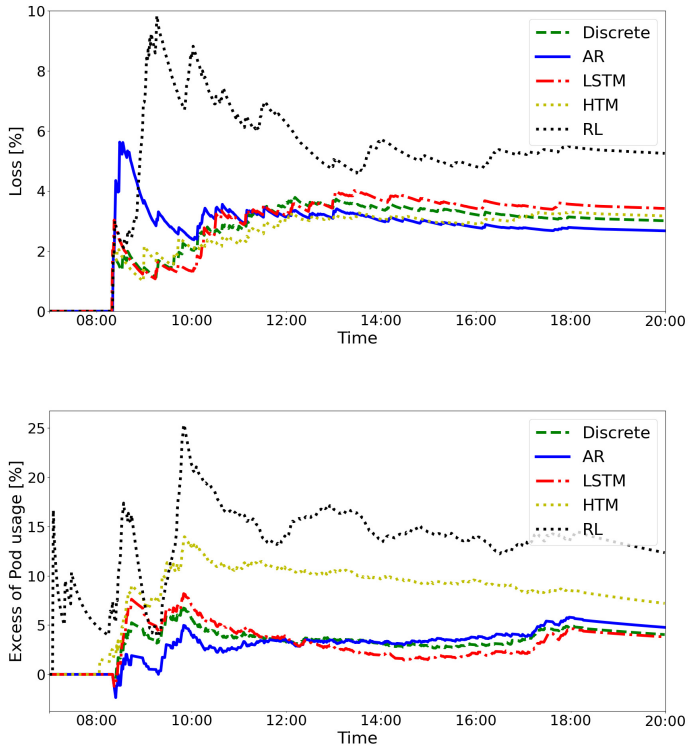


Figure 3.10: Simulations of discrete-time HPA and of the ML-based forecast models

The results of the simulations for all five scaling methods can be seen in Figure 3.10. On the top diagram the cumulative request loss ratio (Loss) is displayed. At the beginning of the simulation, HTM performed best in terms of Loss. In the middle of the day the cumulative loss ratio of the models changed dynamically, and at the end of the day AR finished with the lowest value. RL gave the worst Loss performance throughout the whole day. On the bottom plot the cumulative excess of Pod usage (relative to the oracle) is shown. HTM is far from the optimal resource usage: the y-axis shows that it uses around 8% more Pod minutes than the oracle. The same applies to the RL method: except for a few cases throughout the day, it cannot give better prediction than the rest of the models. The other three methods perform similarly to each other: at the beginning AR is the closest to the optimal, but in the second half of the simulation LSTM gets closer to the optimal. The results show

that although LSTM is closer to the oracle considering the Excess of Pod usage, it leads to higher loss than what is obtained by the other methods. However, those other methods all use slightly more Pods.

We also run the comparative simulations with the synthetic MMPP trace. Results are shown in Figure 3.11: the AR and the LSTM models lead head-to-head in terms of Loss. The hectic MMPP traffic could not be well tracked by the HTM model and therefore it led to overprovisioning. The scaling decisions of RL were delayed or insufficient, thus it resulted in many lost requests again. The lossy discrete HPA model allocated far less resource than necessary, leading to a relatively high amount of losses, so it performed much worse in terms of the number of lost requests than the first two models.

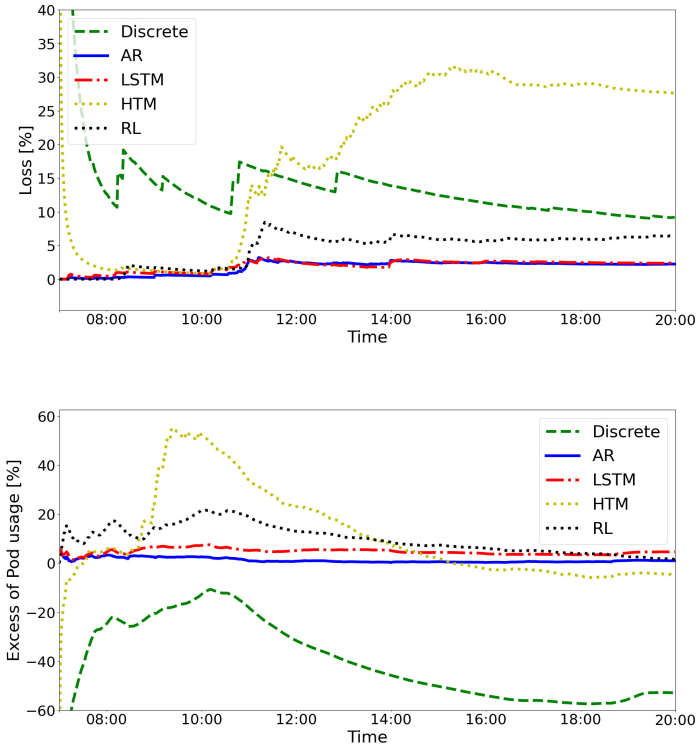


Figure 3.11: Simulations of HPA's discrete-time and of the ML-based forecast models on the generated MMPP traffic

The results demonstrate that there is no single method that would always approximate well the optimal scaling. In fact, as the input traces show varying dynamics throughout the day, methods perform well or poorly episodically. Therefore we design our framework so that models are extendable and system parameters can be changed at any time. An arbitrary number of models can be applied; although more models require more resources, with different models the system can cover different cases of input traffic fluctuations. So we decided to create a scaling engine, in which several methods compete with each other and the active, decision-making method is chosen based on its performance on recent input.

We therefore designed the high level operation of our scaling engine, called HPA+, as follows:

1. it contains four ML-based forecast models that predict that rate of incoming web requests;

2. each minute, it calculates the accuracy of the forecast models for the recent past, and selects the best one;

3. based on the prediction of the selected model, it calculates the number of required Pods (using the profile of the application) for the next minute;

4. if the accuracy of all the ML-based models are poor, it switches back to the default HPA operation temporarily.

Each model's accuracy is calculated based on their past predictions in a brief history window, and the prediction of the best performing model is used for the scaling decision. The evaluation is performed on the average of the Relative Percentage Differences (RPD) in the last n minutes, i.e.,

$$\frac{1}{n}\sum_{i=t-n}^{t-1}\left(2\frac{|\hat{q}_i - q_i|}{\hat{q}_i + q_i}\right), \tag{3.15}$$

where q_i stands for the average QPS in the ith minute and $\hat{q}_i$ is the predicted average QPS for the ith minute. The closer a model's accuracy is to 0, the better its prediction, so from the four ML models, the one with the lowest RPD value will be used each time. We used a parameter for the fallback to HPA, called fallback threshold. If the accuracy value of our models given by (3.15) is greater than this threshold, then the engine switches back to the default HPA operation.

We simulated the operation of HPA+ to examine whether better scaling decisions could be obtained than with HPA, the default auto-scaler in Kubernetes. In order to simulate the racing behavior of HPA+, for different parameter sets, i.e., history window and fallback threshold, we trained the ML models and the HPA discrete model on the 3-day dataset and evaluated them on the fourth day. The active model was determined every minute by (3.15). The number of Pods used and the number of lost requests per minute was calculated considering the scaling decisions of the selected active model only. To account for non-deterministic behavior, we performed this simulation 25 times for each parameter set.

After the parameter search using grid search, we found that a history window of 5 minutes, and 0.3 as fallback threshold should be used in HPA+. In addition to the optimal values of the history window and of the fallback threshold, we also examined how the efficiency of HPA+ would change if the input load was scaled up: we ran simulations in which input request rate was scaled up to 30x of the original trace. We compared the cost induced by HPA+ and by HPA; the cost contains the price of Pod minutes and the penalty for lost requests. We summarize these two cost terms with a weighting factor, called as *trade-off* parameter, which translates the cost of SLA violations, i.e., the number of lost requests, into the cost of cloud resources, i.e., Pod minutes. In Figure 3.12 we show the average ratio of total cost of HPA+ over that of HPA on the y-axis, while the x-axis depicts the scale of the input request rate intensity compared to the original trace. We show the cost ratio for 5 different *trade-off* parameters: 0.01, 0.1, 1, 10, 100. The results suggest that

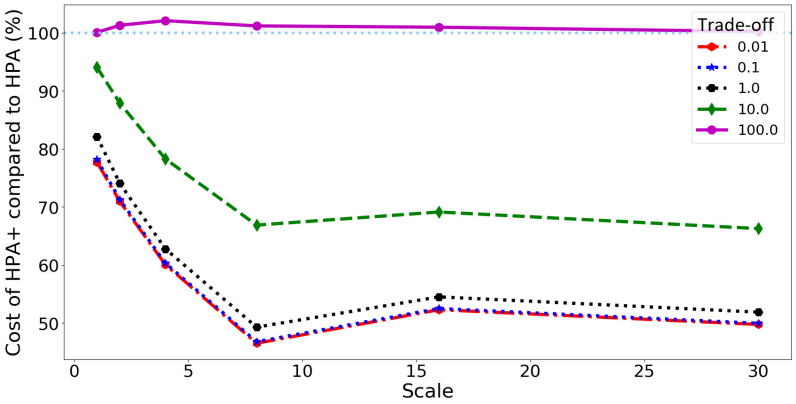


Figure 3.12: Proportion of HPA+ and HPA costs as a function of input load scale

HPA+ incurs slightly higher costs than HPA only if cloud resources are significantly more expensive than SLA violations, i.e., *trade-off* parameter is 100. In other cases HPA+ reaches significant savings, mostly owing to heavily reduced request losses with HPA+. Moreover, HPA+ leads to a significantly cheaper operation when the input request rate is high, i.e., scales over 5x. This phenomenon is due to the granularity of Pods adapted to the request rate: Pods become relatively smaller in capacity compared to the overall demand, hence the number of Pods can be better fitted to the load input.

3.6 Operational model of running microservices

Cloud deployment enables easy scaling to the actual application load. The cost of scaling is greatly determined by the organization of application into scaling units. In this section, we propose an analytical model to reflect the resource footprint overhead at scaling, and the latency overhead of organizing application code into several deployment units. We show that these cost terms are opposing forces that steer the application designer towards organizing the application code in an optimal packaging setup for reaching the sweet spot in terms of operational expenses and application response time performance. On one hand, the co-location of application components within the cloud results in lower operational delays, hence better QoS for the application user. We consider the strictest affinity policy that can be expressed in public clouds today: packaging those components together within a container or a function that must be run on the same hardware. On the other hand, with more packaging comes less modularity, which results in superfluous resource consumption especially during scale-out regimes.

Let us start by modeling the application as a finite set of sub-application parts, called as modules, that can be grouped into separate deployment units, e.g., containers or functions, denoted as scaling units or groups from now on. We assume that any combination of these modules can be packaged together, i.e., in case all of them are grouped together, we arrive back to the monolithic application (from a cloud deployment perspective). We model the modules by their average memory

consumption over a given time period, e.g., an average of 2396 MB memory footprint integrated over 1 hour. We denote those values by $r_1, r_2, \ldots, r_n$. Now, for each of these modules, let us further define a value s that reflects its time-weighted average scaling factor in a steady state operation of the deployed application over the given time period mentioned above, e.g., an average scaling factor of 2.67 due to being scaled to 3 instances for 40 minutes, and having 2 instances for the remaining 20 minutes. The value s_i can be interpreted as the number of invocations of the respective module i of the application which can be run in parallel, i.e., in multiple instances. Application codes that have to run sequentially, i.e., no possible way of parallelism of the module, cannot have a scaling factor s larger than 1.

The problem setup is now translated into dissecting a given application into modules such that each module is packaged separately, e.g., as one or more containers to be grouped in a Pod under Kubernetes, the most widely used CaaS manager, or a function that will be executed in a FaaS platform. Along the process of dissecting a monolithic application into modules, then organizing them into scaling units, the application designer has to focus on the following operational aspects that appear consequently: i) memory footprint of scaling units, ii) overhead of invocation latency between scaling units.

Scaling cost is due to the amount of code that is scaled out unnecessarily in case of executing multiple instances of a scaling unit.

Communication cost An extra latency is added to the application execution time due to invocations across scaling units. Furthermore, when breaking modules and putting them in parallel resources, the increased network latency, as well as reduced reliability, requires careful reasoning about consistency.

We illustrate the scaling cost in Figure 3.13: we depict the modules of an application by 5 rectangles. On the *x-axis* we mark the memory footprint of one instance of each module. The ticks represent 200MB of memory, so the leftmost module's memory footprint is 600MB. We place the modules' memory footprint values right next to each other. On the *y-axis* we denote the number of instances that run in parallel for each module, i.e., the average scaling factor. If there is no parallelization for the modules of a given application, then the height of every rectangle is 1. In the illustrative case shown in Figure 3.13, the scaling factor is 1, 3, 4, 6, and 7 for the individual modules from left to right, respectively.

The overall memory footprint of the whole application is the area under the curve, i.e., the sum of the rectangles' areas. By separating the application into modules and scaling those modules with different factors, the end-to-end application execution time, e.g., response time for a web request, is greatly reduced, but the price to pay is the above-mentioned overhead: inter-module latency. Let us see how the memory footprint changes if some modules are merged into a joint scaling unit. For example, if the application designer decides to add the module in the middle of Figure 3.13 to another module which has either a lower or a higher scaling factor. We assume that the designer does not want to make any compromises on the execution speed at scale out regimes, so in the former case, the applied scaling factor will be the one dictated by the middle module; in the latter case it will be that of the other module. In both cases there will be modules to be scaled to an unnecessary extent,

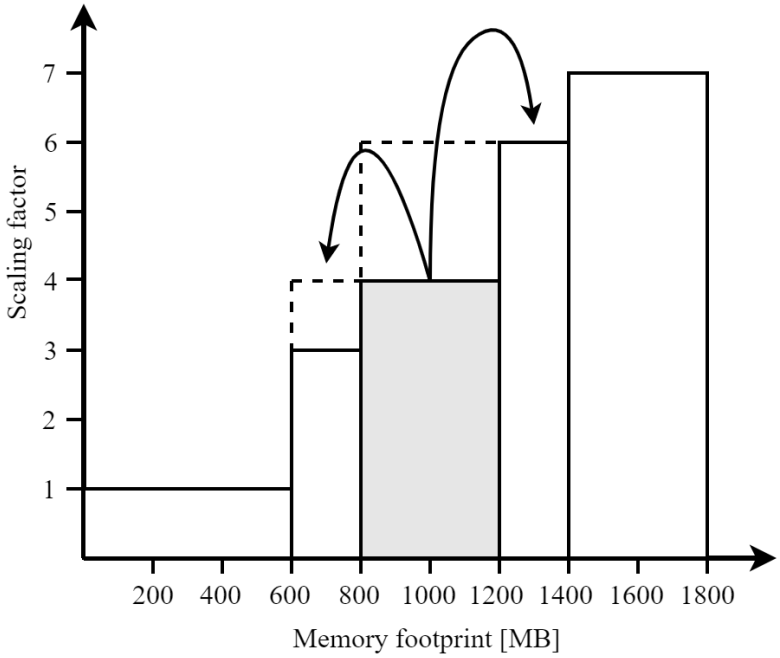


Figure 3.13: Illustration of the scaling overhead model through an example application consisting of 5 modules

leading to an extra scaling cost. In the specific example of Figure 3.13 the extra cost is represented by the dashed rectangles: if the middle module is packaged together with its left neighbor, then the scaling factor of the merged module will be that of the middle module; similarly, merged with its right neighbor, this latter will dictate the scaling factor.

The overall operational cost is therefore increasing by merging different modules of the application that require diverse scaling factors. However, merging them might be necessary to meet the latency requirements dictated by application SLA. The questions naturally arise: how many scaling units should the application designer account for, and which modules should be packaged together into those? We make these statements in the following and provide hints on their proofs.

Lemma 3.12. *For any given number of scaling units, the scaling cost is minimized by grouping the modules together into scaling units following the order of their scaling factors.*

Proof. The proof is indirect. Let us assume an optimal arrangement of modules into scaling units in terms of minimal scaling cost. Without the loss of generality, let the scaling units operate at increasingly ordered $s_1^g, s_2^g, ..., s_x^g$ scaling rates. Furthermore, let module i have s_i scaling rate and belong to scaling unit j with $s_i < s_{j-1}^g < s_j^g$. In this case the scaling cost can be decreased by re-arranging module i (that does not hold the largest scaling factor in its own group j) into another scaling unit for which the scaling factor is higher than its value, but lower than its original group's scaling factor, e.g., into group $j-1$, contradicting the initial assumption about optimal arrangement. □

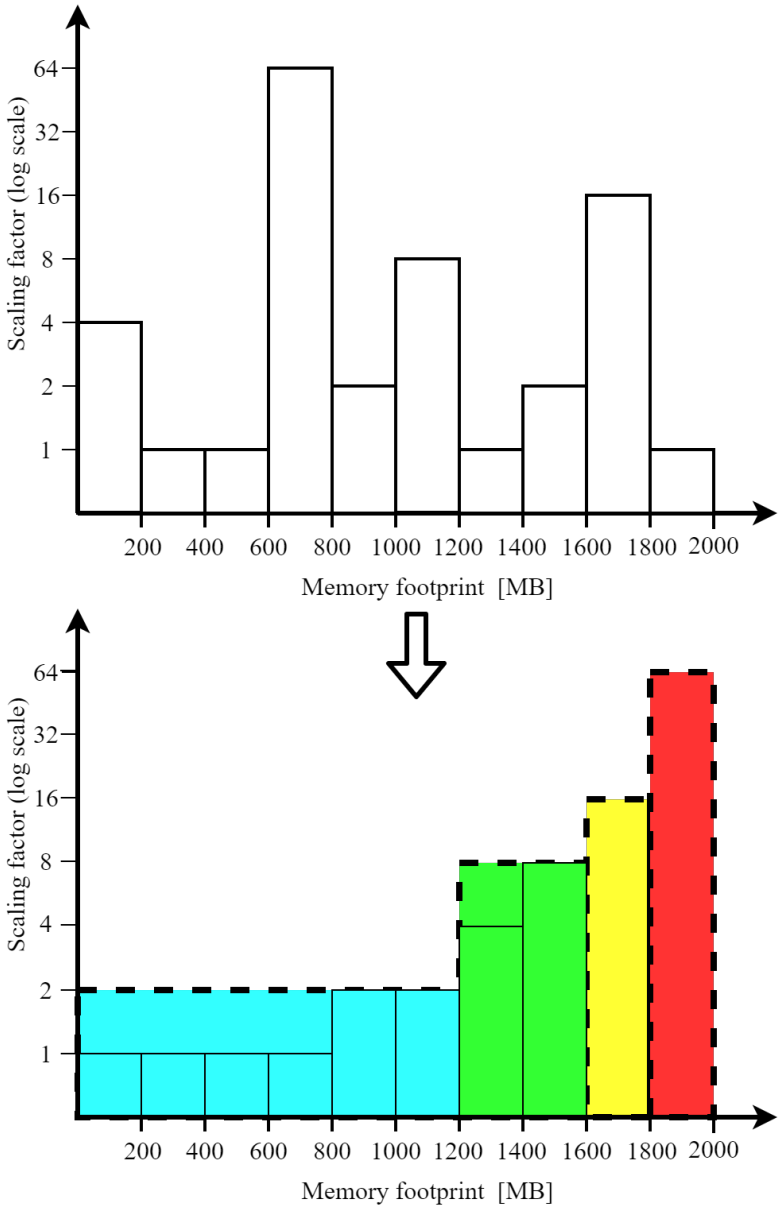


Figure 3.14: Modules of an illustrative example application, ordered by their scaling factor, and grouped (denoted by various colors and dashed line contours)

In Figure 3.14 we depict the modules of an illustrative example application, ordered by their scaling factors and grouped into scaling units along the ordering.

Lemma 3.13. *For any given number (x) of scaling units with minimized scaling cost, for the scaling factor s_b of the modules b on group borders*

$$s_b \geq \frac{s_j^g r_b + s_{j-1}^g \sum_{i \in j-1} r_i}{r_b + \sum_{i \in j-1} r_i} \quad \forall j < x \tag{3.16}$$

holds.

Proof. In an increasing ordered setting of the modules as suggested by Lemma 3.12, the borders of scaling groups are left to such modules b for which the jump in scaling factor is larger than the scaling factor increment (relative to the module's) of the right hand side group multiplied by the width of the module, and divided by the width of the left hand side group. Specifically, assuming x scaling units, for any neighboring scaling unit pair $j-1, j$ for which $1 < j < x$, the following inequality must hold:

$$\left(s_b - s_{j-1}^g\right) \sum_{i \in j-1} r_i \geq \left(s_j^g - s_b\right) r_b. \tag{3.17}$$

It is straightforward to see that in case this inequality does not hold, then the area of the e.g., left hand side rectangle depicted by dashed lines in Figure 3.13, which is expressed by $\left(s_b - s_{j-1}^g\right) \sum_{i \in j-1} r_i$, would be greater than that of the right hand side rectangle, which is equal to $\left(s_j^g - s_b\right) r_b$, resulting in higher scaling cost, thus contradicting with the initial assumption of being at the border of optimal grouping. Equation 3.16 is then derived from Equation 3.17 by rearranging the terms. □

In summary, there is a relatively large jump between the scaling groups' scaling factors in a setting that is optimized for scaling cost. The authors of [72, 73] found that a small percentage of microservices are hot-spots in call graphs, specifically, about 5% of microservices are multiplexed by more than 90% of online services in Alibaba clusters, which creates such large differences between scaling factor values, ideal for marking the borders of scaling groups. For an analytically tractable model, in the next statement the modules are assumed to be infinitesimally small, and the scaling factor is interpreted as a differentiable continuous function over the variable that depicts the cumulative resource demand of the modules sorted in the increasing order of their scaling factors.

Theorem 3.4. *In the continuous model of module resource demand ρ and the scaling factor as its function $\sigma(\rho)$,*

$$\frac{d\sigma}{d\rho}\rho_L + \sigma - \sigma_R = 0 \tag{3.18}$$

must hold for the scaling group borders, i.e., the points on the x-axis that fall on the borders of neighboring scaling groups. ρ_L denotes the width of the scaling group to the left, σ is the scaling factor value that belongs to the scaling group on the left, and σ_R denotes the scaling factor of the scaling group to the right.

Proof. Based on Lemmas 3.12 and 3.13 and the assumption of modules being infinitesimally small in resource demand, $\sigma(\rho)$ is a monotone increasing function and

similarly to Equation 3.17,

$$d\sigma\rho_L \geq (\sigma_R - \sigma)\, d\rho \tag{3.19}$$

also holds, since $\sum_{i \in j-1} r_i$ translates to ρ_L and s^g_{j-1}, s^g_j are denoted as σ, σ_R, respectively. Therefore, the solutions to the given differential equation provide the possible scaling group borders in the proposed continuous model. Equation 3.18 is then derived from Equation 3.19 by rearranging the terms and fixing equality. □

Lemma 3.14. *The scaling cost decreases monotonically when the modules are grouped into more scaling units.*

Proof. The statement holds since any group that consists of at least 2 modules with different scaling factors can be split into 2 groups that have a lower overall scaling cost. As the superfluous resource footprint of the individual scaling units gets smaller, in case of scaling them out, the amount of memory consumption scaled out unnecessarily is also smaller. □

In contrast to the statement in Lemma 3.14, the communication costs increase monotonically with the number of scaling units due to the resource overhead of virtualization and to the higher number of inter-module invocation delays. These opposing effects call for an optimization exercise in order to find the sweet spot in operational costs of polylithic applications.

The limitation of the model is that it ignores the call graph among the modules: it groups those modules together that are close in scaling factor, not necessarily those that frequently invoke each other, or whose lifetime overlaps the most. Therefore, workload affinity is not considered in consolidating software components into scaling units.

3.7 Related work

In this section we present the major achievements in the literature related to cloud management and orchestration. We divide the discussion of the state-of-the-art into parts on i) delay-awareness, high reliability and availability concepts and on ii) auto-scaling methods in cloud environments.

3.7.1 Orchestration of latency-critical cloud-native applications

Latency-sensitive and data-intensive applications, such as IoT or mobile services, are leveraged by edge computing, which extends the cloud ecosystem with distributed computational resources in proximity to data providers and consumers. This brings significant benefits in terms of lower latency and higher bandwidth. However, by definition, edge computing has limited resources with respect to cloud counterparts; thus, there exists a trade-off between proximity to users and resource utilization. Moreover, service availability is a significant concern at the edge of the network, where extensive support systems as in cloud data centers are not usually present.

To overcome these limitations, [12] proposes a score-based edge service scheduling algorithm that evaluates network, compute, and reliability capabilities of edge nodes. The algorithm outputs the maximum scoring mapping between resources and services with regard to critical aspects of service quality. [53] introduces a new platform for enabling an edge infrastructure according to a disaggregated distributed cloud architecture and an opportunistic model based on bare-metal providers. Results from a multi-server online gaming application deployed in a real geo-distributed edge infrastructure show the feasibility, performance and cost efficiency of the solution.

In order to meet the rapidly changing requirements of the cloud-native dynamic execution environment, without human support and without the need to continually improve one's skills, autonomic features need to be added. Embracing automation at every layer of performance management enables us to reduce costs while improving outcomes. [65] lists the definition of autonomic management requirements of cloud-native applications, and the authors propose that the automation is achieved via high-level policies, while autonomy features are accomplished via the rule engine support. One such feature of online scheduling in a cloud-native context is migration. A large body of research has tackled the issues around migration of VMs, containers, etc. in the cloud. E.g., [57] proposes an energy-aware VM migration technique for cloud computing. The proposed technique migrates the maximally loaded VM to the least loaded active node while maintaining the performance and energy efficiency of the data centers.

In the era of cloud services, there is a strong desire to improve the elasticity and reliability of applications in the cloud. The standard way of achieving these goals is to decouple the life-cycle of important application states from the life-cycle of individual application instances: states, and data in general, are written to and read from cloud databases, deployed close to the application code. Rooted in cloud-native computing, the stateless design outsources the state embedded in computing entities, e.g., VMs, containers, Pods, virtual network functions, to a dedicated state storage layer, facilitating elastic scaling and resiliency [114]. In [114] the authors propose a system design that can be adapted to any cloud application without the need for complex coordination among the network control, the stateless application elements, and the state storage backend. They present the first product-phase realization of the stateless paradigm, an operational virtualized IP Multimedia Subsystem that can restore the live call records of thousands of mobile subscribers under a couple of seconds with half the resources required by a traditional "stateful" design.

The high performance requirements on the application impose strict latency limits on these cloud storage solutions for state access. Cloud database instances are therefore distributed on multiple hosts in order to strive to ensure data locality for all applications. However, the shared nature of certain states, and the inevitable dynamics of the application workload necessarily lead to inter-host data access within the data center (or even across data centers, if the application requires a multi-data center setup). In order to minimize the inter-host communication due to state externalization, the authors of [114, 115] propose an advanced cloud scheduling algorithm that places applications' states across the hosts of a data center. In such a cloud-native setting, stateless cloud applications and an adaptively self-synchronizing distributed cloud database alleviate the long-standing issues of live migration within the cloud.

Several research papers have been published that all propose some kind of scheme for improving the availability and reliability of applications in the inherently untrustworthy context of edge cloud infrastructure. [58] tackled the problem of separate software stacks between the edge and the cloud with no unified fault-tolerant management, which hinders dynamic relocation of data processing. In such systems, the data must also be preserved from being corrupted or duplicated in the case of intermittent long-distance network connectivity issues, malicious harming of edge devices, or other hostile environments. A self-adapting scheme is proposed in [109] which uses static and dynamic backups for VNFs over both the edge and the cloud in order to provide high availability. Fan. et al. [35] propose a framework to provision availability of SFC requests in a data center. None of these research works consider multiplexing backup resources for multiple virtual instances like our scheduler does. Yala et al. [139] propose a solution for their optimization problem that strive to optimize the trade-off between availability and latency. However, their work deploys VNFs without deploying backup resources.

Although, the solution of Kanizo et al. [61] and Raba [144] both multiplex backup resources, they ensure high availability for VNFs with dedicated backup nodes. In contrast, our scheduler performs resource provisioning on general nodes that contains Pods as well. In [36, 19] the authors consider the replica and virtual function placement to achieve lower migration time, however they did not consider minimizing the provisioned resources assigned to replicas. Authors of [140] investigate the fog resource provisioning problem for deadline-driven IoT services to minimize the cost considering the probability of resource failures. They assume that VM failures are temporary and recoverable. In contrast, we argue that each node's failure should be prepared for.

An online resource orchestration algorithm which takes into account network aspects is proposed in [43]. The algorithm enables the orchestrator of OpenStack to manage a distributed cloud-fog infrastructure. An embedding algorithm is proposed in [90], which instantly deploys end-to-end delay-constrained services while applying a cost-aware VNF migration strategy. The authors' hybrid orchestration approach unites the advantages of online heuristics and offline optimization in their service orchestration method, with the goal of providing fast service placement and minimizing the cost due to VNF migrations.

The research community has already started extending the Kubernetes scheduler to support edge computing architectures with network awareness [23, 84, 107]. In contrast to our work, the scheduling method in [23] does not take into account the delay directly between the edge nodes. Authors of [84] propose a content delivery method that improves Kubernetes scheduler with awareness to network distance using the Autonomous System path of the Border Gateway Protocol. In contrast, we use the measured delay values as the network distance property. An extension, called Network-Aware Scheduler, is implemented in [107] enabling Kubernetes to make resource provisioning decisions based on the network infrastructure properties like latency and bandwidth. Although, the application requests in [107, 84] do not define latency requirement that has to be met during their scheduling. Furthermore, none of the previous works consider providing high reliability for the applications, and dynamic topology clustering to relax the difficulties that a large scale architecture

poses. We argue that the main goal of edge computing, i.e., hosting delay critical applications, must be aware of not only network latency, but it must also take into account the unreliability and the large number of edge nodes.

3.7.2 Auto-scaling solutions in the cloud

Since elasticity and dynamism are key concepts to cloud computing, offering appropriate application scaling is one of the most important features for a cloud provider. [71, 3] give comprehensive surveys about scaling solutions applied in data centers. They categorize auto-scalers according to their underlying theoretical models. We follow suit, and highlight those recent works that fall close to our proposed solution.

Threshold-based: Authors of [2, 101] improved vertical elasticity in cloud systems of lightweight virtualization technologies with threshold-based scaling rules. Authors of [2] proposed ElasticDocker which supports vertical elasticity of Docker containers based on the IBM's autonomic computing MAPE-K principles. In [101] the authors presented the Resource Utilization Based Autoscaling System, which improved Kubernetes' Vertical Pod Autoscaler with the ability of dynamically adjusting the allocation of containers non-disruptively in a Kubernetes cluster. Both papers incorporated container migration and examined the possibilities in vertical scaling, while our proposed solution improves the horizontal auto-scaler. Khazaei et al. [62] presented the architecture of Elascale that provided auto-scalability and monitoring-as-a-service for any type of cloud software system, making scaling decisions based on an adjustable linear combination of CPU, memory, and network utilization. We argue that this simplification is a limiting factor in the scaling performance.

Queuing model-based: Several queuing models describe scaling systems by assuming a memoryless arrival process and an adaptive number of servers. Kaboudan et al. [59] presented a discrete-time queuing model with a dynamic number of servers using a threshold-based scaling policy. They ran simulations, and compared their model to the behavior of an M/M/c queue. Mazalov et al. [77] proposed a queuing model with an on-demand number of servers, which depended on the length of the queue, and they assumed a Poisson arrival process with a specified rate. Jia et al. [48] introduced a queuing model which used a threshold-based policy to better describe an industrial production process. The model assumed a MMPP as arrival, and the number of servers was selected from two predefined values based on the queue length. In all these cited papers we see the Markovian assumption of exponential inter-arrival times to be a limiting factor. Nevertheless, we also use an MMPP/M/c model in our analytic approximation of HPA.

Control theory-based: Using control theory, the behavior of a dynamic system is changed by examining the output and reference values: the goal of the controller is to align the actual output to the reference based on the feedback to the input system. In an auto-scaling context, the reference value is the targeted SLA, and the output values to evaluate performance come from the system, e.g., CPU load or response time. For maximizing application QoS, while meeting both a time constraint and a resource budget, Zhu and Agrawal [146] developed a framework with Proportional-Integral control and Reinforcement Learning (RL). They showed that

their solution can efficiently scale the VMs under the application with low overhead. Ali-Eldin et al. proposed two solutions for horizontal cloud elasticity: in their first work [5], the infrastructure was modeled as a G/G/N queue with variable N, which was used to design a reactive-adaptive proportional controller that acted based on the load dynamics, and could respond to sudden changes in it. They also managed to avoid oscillations and premature resource reduction; in their second article [6], two adaptive hybrid controllers were introduced that used both reactive and proactive control to change the number of VMs allocated to a service. Their system was analysed using different deployment scenarios with several real life workloads, and the proactive and reactive controller based scaling could outperform both the regression and the fully reactive scaling. Kalyvianaki et al. [60] used a Kalman filter to adjust the CPU allocation based on past utilization observations. The proposed system is able to scale multi-tier applications and has the ability to configure itself to any workload conditions. Baresi et al. [14] proposed a discrete-time feedback controller to scale resources both at VM-, and at container-level, and has the advantage of handling microservices-based applications. Their experiments showed that the proposed controller can outperform Amazon's Autoscaling significantly. Control theory-based models show high efficiency in application scaling. Several models combine other types of scaling methods, e.g., RL, performance or demand prediction inside the controller to adapt to the application needs. Another advantage of these solutions is that the controller can operate in the order of seconds as presented in [60], thus the scaling algorithm can respond quickly to changes in resource usage.

Reinforcement learning-based: Arabnejad et al. [11] combined two RL-based approaches: Q-learning and state-action-reward-state-action algorithms with a self-adaptive fuzzy logic controller that drove dynamic resource allocation for VMs. Horovitz et al. [51] presented a threshold-based solution for horizontal container auto-scaling that used Q-learning to tune the scaling thresholds. Rossi et al. [105] proposed RL solutions (i.e., Q-learning, Dyna-Q, and model-based) in Docker Swarm that exploited different degrees of knowledge about system dynamics. In our autoscaler engine we also leverage the power of RL methods.

Performance prediction-based: Wajahat et al. [130] proposed a NN-, and regression-based application agnostic autoscaling solution, called MLscale. They used a multi-layer feed-forward NN to build the performance model of the application based on the request rate and system resource statistics. The model was able to predict the response time of the application. To predict the new response time after a proposed scaling action, they trained multiple linear regression models, which were able to determine the metrics from the current state taking into account the scaling decision. Rahman et al. [98] proposed a solution to predict end-to-end tail latency of microservice-based applications and to scale based on that. They trained several ML methods, like random forest, linear regression, support vector regression and deep NNs, to predict the latency of the application. For a given workload condition, they used the best model to find the highest resource utilization values of the relevant microservices, at which the given SLA targets would not be violated, and the scaling threshold was set to the found resource utilization value. Authors of [143] presented Microscaler, an autoscaling system that automatically detected SLA violations, determined the services requiring scaling, and evaluated how much resource those needed. They introduced Service Power, a metric that could be cal-

culated from the response times of the service, and they used this metric to find the services which needed scaling. To determine the required instances for these services, they used Bayesian optimization and a step-by-step heuristic approach. Authors of [106] provided a solution called Autopilot, an ML-based horizontal and vertical scaler used by Google to configure the resources for tasks in a job. The primary goal was to reduce the difference between the limit and the actual resource usage (slack), while minimizing the risk that a task is killed. They used several ML techniques to predict the vertical resource limits based on historical data, and different ruled based approaches to horizontal scaling. In practice, Autopiloted jobs showed a slack of 23%, compared with 46% for manually-managed jobs.

Demand forecast-based: Short-term demand forecasting has been in the focus of researchers of various application domains for many years. In the energy sector, it is very important to know the power generated by wind turbines in advance. To solve this problem Li et al. [68] developed a 4-input NN which turned out to be better than the single parameter traditional approach. Catalao et al. [21] proposed a 3-layered feed-forward NN approach to forecast next-week electricity market prices. Their approach proved to be better than previously presented auto-regressive integrated moving average model (ARIMA) methods for the reason that it is less time consuming and easier to implement. For cloud computing, Chen et al. [22] implemented a dynamic server provisioning technique to minimize the energy consumption of data centers. A custom AR method was presented in their study to forecast the number of connections on Windows Live Messenger servers. In [39] the authors applied signal processing and statistical learning algorithms to achieve online predictions of dynamic application resource requirements. Their forecast solution was based either on signatures of historic resource usage, or on a discrete-time Markov chain with a finite number of states, depending on whether the input traces showed repeating patterns or not. An effective prediction model was also provided by Islam et al. [55] for adaptive resource provisioning in the cloud. They evaluated several ML models, such as NNs and linear regression, on a dataset. They claimed that their solution was suitable for forecasting resource demand ahead of the VM instance setup time. The authors of [79] proposed an adaptive prediction method using genetic algorithms to combine time-series forecasting models, such as ARIMA, simple and extended exponential smoothing.

Many of the listed approaches modeled the resource needs of the application using ML techniques. In order to avoid both excessive over-provisioning and SLA violations, we use such performance models, and we also follow the ideas behind the line of work of demand forecast-based methods.

Chapter 4

Bandwidth allocation in access networks towards the cloud

4.1 Introduction

The current state in the evolution of Internet is the "wireless Internet" in which Internet access has become available for anybody, anywhere at any time via mobile devices. Quality of experience in wireless access networks has been and, with the proliferation of the Networked Society, will be an important factor in telecommunications. In this chapter we propose service quality assurance frameworks with which the existing tools in the hand of network operators are extended with the capability of *user-driven* or cloud-based quality control. In the first case the users get an opportunity to signal their online demand for scarce resources towards the network, which in turn can improve its decisions on resource allocation with the ultimate goal of raising the satisfaction of its users. As a specific example, we make the case of user signals for urgent bandwidth demands, and of the scheduling decisions made at the access point based on those. In the second case we present a cloud-based IoT architecture that maximizes utility when high-bandwidth video streams have to share a narrow uplink channel. We propose a resource allocation method that benefit of the context-based prediction engine. This allows for a significant reduction in uplink bandwidth and processing power in the cloud, and creates an efficient multiplexing among the resources used to detect events in video streams in parallel.

This chapter is organized as follows. First, we derive the model of a distributed framework with the elements of stochastic game theory. Our model describes the state of the access network via the users' bandwidth demand, their backlogged jobs, and the strategies they can choose with the aim of reaching a desirable bandwidth allocation. Second, we propose a central allocation scheme for the uplink shared by multiple video streams of e.g., an IoT system. We exploit the benefits of predictable events for which data to be uploaded to the cloud can be compressed according to the optimization target of the system operator.

4.2 An uncoordinated bandwidth sharing model and its analysis

In this section we first present our framework, engineered to solve the aforementioned issues, then we build its simplified model, and give an analytical formulation and an example for solving the problem of finding the optimal policy.

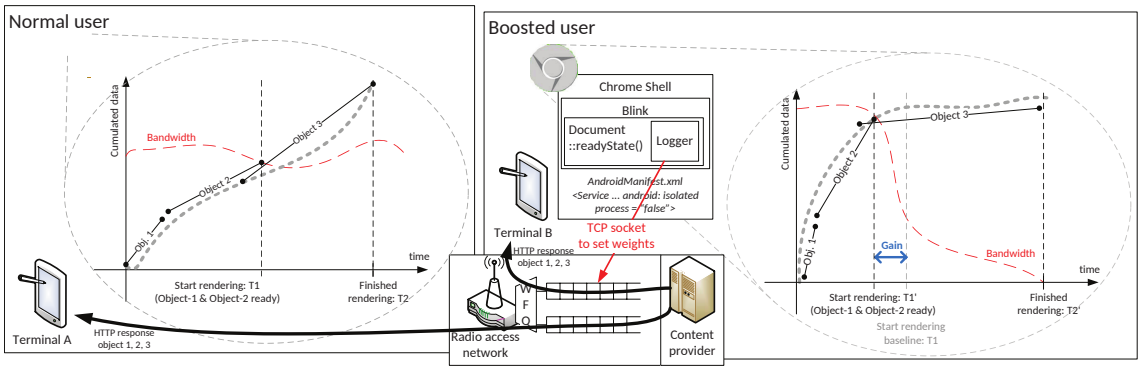


Figure 4.1: The Boosting Framework in a toy example for 2 users

In Figure 4.1 we depict the components of our framework, i.e., the radio access point, two terminals and a content provider server, with a toy example that introduces the notion of "boosting" in a case of e.g., web page rendering. The prioritizing (or boosting) logic in the wireless access point is stylized with different queues and a Weighted Fair Queuing (WFQ) scheduler [95]. The "Normal user" with Terminal A is not using the boosting service, while the "Boosted user" with Terminal B indeed does. The plots drawn at both users show time on their x axes, and the accumulated downloaded data and the received bandwidth on their y axes with dotted and dashed lines respectively. Solid sections show the web objects that are downloaded in the respective bandwidth-time products.

As the result of boosting, the toy example shows that although the total time to download all 3 web objects is the same for the 2 users, the "Boosted user" gets hold of the first 2 objects earlier than the "Normal user", which is proved to be critical in terms of QoS, hence the benefits of using our proposed Boosting Framework. While the download bandwidth for the "Normal user" remains at the same level throughout the download session, the "Boosted user" receives higher bandwidth allocation at the beginning, and lower at the end. The boosting service in this case is implemented in the wireless access point, and is triggered by the web browser's custom plugin which sets the weights higher in the beginning and lower at the end compared to the default value for the WFQ scheduler in the wireless access point. In the following we model this as submitting bids to an auction where boosting bandwidth is allocated to users for time slots.

We split the time horizon into uniform slots and assume that in a given time slot only one user can get a unit amount of boosting bandwidth, resulting in the fact that one unit of the user's jobs gets boosted. This way we discretize our model and make sure that the auction in which the wireless access point selects the user

whose job will be boosted is a single-unit auction. We apply a second-price auction: highest bid wins and pays the second highest bid for the single item. The usual notions and notations related to our auction are listed below:

- Users are denoted as $\mathcal{I} = \{1, 2, \ldots, i, \ldots, n\}$.
- The amount of boosted jobs of user i in a given time slot t is denoted by $x_i(t) \in \{0, 1\}\ \forall i$ such that $\sum_i x_i(t) = 1\ \forall t$. The amount of boosted jobs is defined in terms of traffic volume (given by the product of time slot length and the amount of bandwidth allocated for boosting). In each time slot only one user wins the opportunity to get its jobs boosted.
- Stochastic demand: $d_i(t)$, job arrival events are linked to time slots and job sizes are defined in the aforementioned boosted traffic volume units.
- Jobs that are not boosted accumulate in the backlog of the respective user, denoted as J_i for user i. We denote a job with j_i in the backlog $J_i = \{j_i\}$ of user i, its size as $|j_i|$ and its age, i.e., the time it spent in backlog, with $\tilde{j_i}$.
- The value of getting a boosting opportunity is $u_i(t) = u_i(x_i(t), J_i(t)),\ \forall i$, i.e., all the backlogged jobs influence the received utility because by not getting the opportunity all those jobs get delayed, hence negative effect on QoS.
- Users bid for boosting bandwidth with $b_i(t)$ in the discrete time slots. In a system implementation the bids are best produced by a browser plugin, just as it is suggested in the toy example of Section 4.2.
- The second highest bid, i.e., the price to be paid, is denoted by $c_i(t)$. The budget of user i in time slot t is $m_i(t)$. Accounting user budgets and subtracting costs to be paid at auctions are best handled by the access point in a possible system implementation.

The system behaves as follows. In each time slot the system is in one of the states $\mathcal{S}$ that describe *traffic backlog to boost* and *remaining budget* for each user. If there are less jobs to be boosted than what the system can handle in a time slot, then naturally all of them get boosted. On the other hand, when the resource demand exceeds the offer, i.e., there are more jobs waiting for being boosted in the system than what can be served, a job ends up either being boosted or staying in backlog by the end of the time slot.

The aim of the analysis is to find the stationary policy that optimizes the service allocation according to various Key Performance Indicator (KPI)s. The possible state transitions from one time slot to the subsequent one are due to job arrivals and serving jobs, which latter is driven by the job backlogs via boosting attempts. A Markov decision process based policy optimization approach would require the definition of selected actions $\pi_i(b_i|s) = \mathbb{P}\left[A_i^t = b_i | S_t = s\right]\ \forall t$, where A_i^t denotes the random variable depicting user i's action in time slot t when the system is in state s. Due to the high memory dependence of the system behavior the MDP based analysis is infeasible.

Instead of state space based optimization, we treat the problem as searching for a policy in a bid-based stochastic game. In this setting the jobs are generated at users randomly, then they submit bids, finally one unit of the winner's backlogged jobs gets served. The auction winning user's job gets boosted, others' jobs to be boosted remain in their backlogs.

The payoff function users optimize is defined as follows.

Definition 4.1. The user payoff is equal to the sum of utilities of jobs served minus the cost paid for the service, i.e., for user i and time T: $p_i(T) = \sum_{t=1}^{T} u_i(t) - c_i(t)$.

Although the value of boosted jobs is what directly affects the user QoS, we also integrate the cost that a user has to cover from its centrally allocated budget into the payoff. Doing so we intend to make the payoff resonate with the repeated and stochastic character of the game: this creates an incentive not to use up all the allocated funds at one, but keep savings for periods when multiple jobs arrive.

We assume that the rational users strive to maximize their payoffs, and therefore seek the optimal strategy: $b_i^* = \arg\max_{b_i} \lim_{T\to\infty} \frac{p_i(T)}{T}$ with budget constraint $\sum_{t=1}^{T} m_i(t) - c_i(t) \geq 0,\ \forall T$. The strategy (bidding policy) of the user allows for describing a wide range of dynamic user and system behavior. Therefore optimization of various KPIs can be implemented through these user strategies.

In order to demonstrate the complexity and the flexibility of the model, here we show a specific example. We assume a game of 2 players, unit-size jobs, service capability of 1 job per time slot in total and for both players a utility decreasing with the age of the head job of the player's backlog: $u_i(t) = u - a_i(t)$. For the sake of simplicity we assume that a player can choose between 2 actions in each time slot: either bids with the actual utility of its head job, or bids with zero.

In Figure 4.2 we demonstrate two cases in which both players have one job, the first player's head job is i time slots old, the other player's head job is j time slots old. The left-hand side shows the case in which the players adopt the policy in which they bid with the head job utility, the right-hand side graph shows the case for zero value bids. The circles represent states and only those are shown that will be reached from the one-one job state until ending up in an empty system (no new job arrivals are supposed). In the circles we depict the number of jobs for the two players. On the arrows, we show the payoff of the player that wins the auction between the two states.

In the first case, whoever has the more recent job (e.g., $i < j << u$) will win the auction, but has to pay the other player's utility as cost (e.g., $p_i = u - i - (u - j) = j - i$). Then the other player can get its job boosted without any cost (e.g., $p_j = u - j - 1$), but with less utility, as the head job got older. In the second case the first time slot gets auctioned to a randomly selected player with no cost, hence e.g., $p_i = u - i$, and the other player gets the second time slot for no cost, as in the first case. Based on these payoffs, as long as $j - i > 0.5(u - i) + 0.5(u - i - 1)$ stands, i.e., $u > j > u - 0.5$, the first player is better off with the utility-based bidding policy. Otherwise, and this is the more probable case of the variables, zero value bidding results in higher payoffs. Furthermore, if $j - i < u - i - 1$, i.e., $j < u - 1$, it is actually more profitable for the first player to let the second player, with the

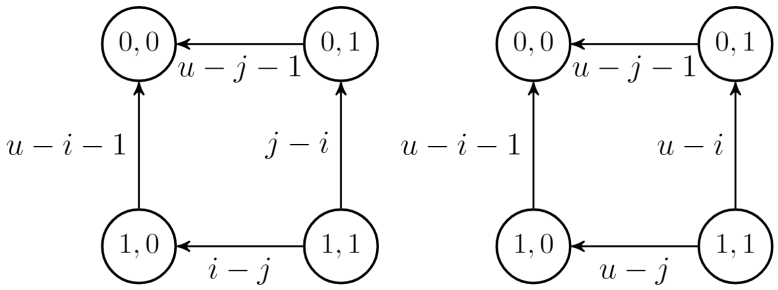


Figure 4.2: Toy example for utility-based bidding policy (left-hand side) and zero bidding policy (right-hand side)

older head job, win. The lack of costs compensates the player for the loss of utility in the latter time slot.

Note that this toy example with artificial payoff assumes that no new jobs arrive before the second allocation, and we assume that players either bid with their actual head job utility or with zero. These are restrictive assumptions, but keep the example tractable and show that even without the stochastic element of job arrivals how the sequential (or repeated) nature of the game rules out the utility-based strategy from the set of dominant strategies in these second-price auctions.

Hindered by the analytical complexity, we present a numerical evaluation of our model, and we show simulation results for different heuristic policies. Our main assumption is that urgent bandwidth demands, if not served with the required resource allocation, lose their valuation with time: jobs are depreciated in the backlog. Our goal is to show that despite this pushing time constraint even a user-driven resource allocation framework may alleviate congestion situations.

Here we present its parameter settings, the analysis we made the cases for and finally the results we obtained. We are interested in the interplay of different bidding policies. In our simulations we assume that users switch among our heuristic bidding strategies following an evolutionary process, i.e., moving towards policies that provide higher payoff. First we refine the valuation of boosting for which the users bid in each round.

Definition 4.2. The value of boosting a job of user i is $\max(u - \epsilon t, 0)$, i.e., the initial utility is linearly diminishing with the rounds spent in backlog.

Before introducing the policies we investigated, let us define the term *opportunity cost*.

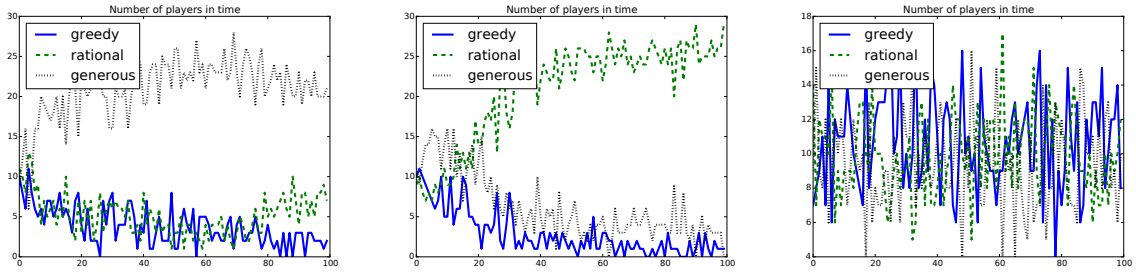
Definition 4.3. The opportunity cost is the future loss of valuation of backlogged jobs: if a user does not get the boosting bandwidth in a given time slot, the valuation of its backlogged jobs decreases by the next time slot. Therefore the opportunity cost for user i with jobs $J_i = \{j_i\}$ is $\sum_{j_i \in J_i} |j_i| \epsilon$ where $|j_i|$ is the size of the job j_i still in backlog and ϵ is the depreciation of backlogged jobs from Def. 4.2.

Now, given the time-sensitive job utility and the opportunity cost, we define three heuristic bidding policies.

Definition 4.4. With *greedy* policy the player bids its whole budget; with *rational* policy one bids the actual utility of the jobs to be boosted; and with *generous* policy one bids the opportunity cost (Def. 4.3). In all policies the player's budget is the upper limit of the bid.

We run simulations with the following parameters in order to demonstrate the pros and cons of the proposed heuristic policies, and to compare the distributed auction-based allocation with traditional bandwidth sharing.

- Number of users: $|\mathcal{I}| = 30$.
- Jobs are generated at users with independent and identically distributed exponential random inter-arrival times having mean $\beta = \frac{1}{\lambda} = |\mathcal{I}|$ (Poisson process), and all the jobs are unit-sized.
- The age of a job j_i of user i is given by the number of rounds the job has spent in the backlog, i.e., a job's age is 0 in the round it arrived, 1 in the following, and so on.
- Jobs in the backlog are continuously served with the bandwidth not allocated for boosting, therefore in each time slot their size decreases by $\delta = 0.2$. This however does not induce any valuation for the user.
- Users bid for a bandwidth-timeslot unit in each round. For the auction winner i $x_i = 1$ in the given round and $x_j = 0$ $\forall j \in \mathcal{I} \setminus i$. This traffic opportunity is used to boost jobs in the winner's backlog under FIFO policy.
- Each user gets the same budget increment in each round that they use for bidding. Users strive to increase their payoffs which is the value of boosted jobs minus the cost of winning the auction.
- We assume the same initial utility and depreciation for all jobs and all players: $u = 1$ and $\epsilon = 0.1u$.



(a) $\frac{0.1}{30}$ job arrival rate **(b)** $\frac{1}{30}$ job arrival rate **(c)** $\frac{10}{30}$ job arrival rate

Figure 4.3: Evolution of number of players of various strategies

It is well-known that the *rational* policy, i.e., truthful bidding, would be the optimal strategy in case the game was a one-shot game because of the desirable characteristics of second-price auctions. We make the case, however, for a stochastic game, an

extended game with stochastic transitions between different states represented by the job backlogs.

In Figure 4.3 we depict the number of players implementing each policy in systems with increasing load levels. We let users change their applied bidding strategy mimicking the dynamics of evolutionary game theoretical models: we assume that when a given user wants to bid for boosting jobs it randomly selects one of the policies with probabilities proportional to the average accumulated payoffs of users grouped by their actual strategies. In the beginning users are evenly split among the policies to start with. The job arrival rate is 0.003 in Figure 4.3a, 0.03 in Figure 4.3b and 0.3 in Figure 4.3c with a number of players of 30 in all cases. The solid, dashed and dotted lines show the evolution of player counts with the 3 presented policies on the y-axis in the function of simulation rounds on the x-axis. When the load is low (Figure 4.3a) the *generous* policy prevails, when the system is saturated (Figure 4.3b) the *rational* policy seems to provide the highest payoff, while in an overloaded system (Figure 4.3c) no policy is better than the others. The *generous* bidding strategy pays off on the long run when the average system load is low because in most cases the relatively low bid is enough to get jobs boosted, and the budget is saved for bursty times. Intuitively, when the system is close to its saturation it is worth bidding with the actual value of the jobs to-be-boosted (*rational policy*) in order to beat users applying other bidding strategies and to get jobs boosted as soon as possible. In any case it is not beneficial to burn the whole budget in single bids applying the *greedy* policy, unless the system is overloaded for a long time (Figure 4.3c) but then no policy beats the others at providing the user better chance to get a boosting opportunity.

4.3 A coordinated cloud resource optimizer method

For a cloud-based data intensive system two constraints are to be met: low latency for real-time applications and affordable uplink speed. In this section we investigate the problem of connecting an IoT system to the cloud with the aim of processing time-critical high bandwidth data. Our main idea is to make such systems resource-efficient by allowing the cloud to control the rate of data streams sent for processing from the site to the cloud. Roughly speaking the cloud predicts the location and timing of the next possible events that need to be captured. Based on the prediction some video streams are awarded more bandwidth and other cameras are switched off or instructed to operate at lower bandwidth settings. In this way the uplink is mostly occupied with data important for the analysis.

Figure 4.4 shows the functional blocks of our system. Low data rate *sensors* provide a steady stream of raw data that is processed in the cloud for contextual information. These components are not affected by the resource allocation algorithm, they are considered a constant operational cost. Within the cloud there are *detector* blocks that are responsible for processing the camera streams. These perform the most resource-intensive task: detecting events from video data. Also running in the cloud, there are processes that implement the *predictor* algorithms that forecast

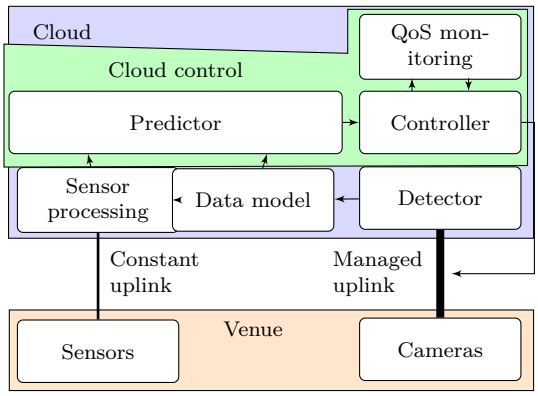


Figure 4.4: System overview with functional blocks

future events with some probabilistic confidence based on a dynamically recalculated *data model*. In the core of the proposed system, we have the *controller* block which, based on the different predictions, manages uplink bandwidth and CPU resources, i.e., performs real-time optimization of resources. The resource allocation is a synchronous process, e.g., discrete time increments of 33ms can be based on the frame rate of the installed cameras, and resource allocation decisions are made and enforced for each time frame. Long term utility maximization is handled by the *QoS monitoring* component that ensures overall fairness and possibly service level differentiation.

Our contribution is the resource allocation algorithm that maximizes the overall utility of aggregated processing of IoT data in the cloud. In particular we are focusing on *how to handle situation when there are not enough resources*, e.g., if the uplink does not have sufficient bandwidth for traditional multiplexing of video streams coming from many sources, and the *controller* must make fast choices between the data stream requests. We investigate how an optimal allocation can be reached. In this section we present our resource allocation scheme, we provide the definition of the optimization problem and we show a fast dynamic programming algorithm to compute the optimal solution.

4.3.1 System model and the resource allocation problem

For simplicity we first assume that the cloud has infinite capacity and the only bottleneck is the uplink bandwidth. Later we explain how to extend the formulas for multiple resources such as CPU core, GPU core, memory, disk, etc. Our aim is to maximize QoS, which is the number of detected events of interest, where the event detection accuracy depends on the amount of resources we allocate to the detection module.

The proposed system provides a resource-efficient (e.g. energy) control mechanism for IoT systems by adaptively orchestrating all the underlying elements of the architecture, such as sensors, agents, actuators, etc. The system comprises of a *detector* module that keeps track of the state of the system, a *predictor* module that can be used to forecast certain schedules, and a *controller* module that executes the orchestration logic and issues configuration settings and commands to the elements of

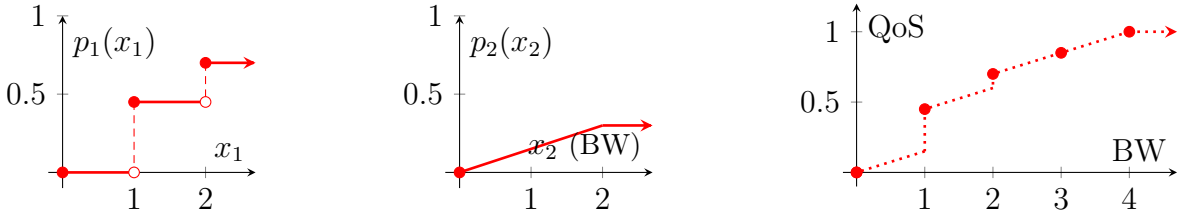
the system, therefore implements complex resource allocation. The proposed methods are applied in the controller module with the goal of operating the underlying IoT system with the highest achievable precision while keeping the overall resource consumption to the minimum.

The different aspects that the *controller* module takes into account when making decisions for the underlying system are the following.

1. The uplink capacity to the cloud, which is usually a scarce resource from the sensor side.
2. The sensor sources and resolution, e.g., image resolution in case of a camera, sampling rate in case of a temperature sensor.
3. The reserved computing power, which greatly influences the precision of both event detection.

All these aspects and constraints interplay in the *controller* module that can be implemented to maximize QoS given the resource (also cost) constraints at all times with high frequency. In order to do this, we propose to implement fast algorithms, e.g., dynamic programming, with discrete target functions.

The cloud runs the *detector* process to capture the events of interest. In general, the accuracy of event detection depends on the quality and quantity of the received data and the allocated cloud resources. Our goal is to minimize the number of missed events, or the price of missing events if prices to event misses are assigned. We discretize time into time frames, and at the end of each time frame the *predictor* process computes the value of a *utility function*, which is the expected number of events detected in the next time frame depending on the amount of resources it will receive (see Figure 4.5 as an example). The *controller* aggregates these utility functions to maximize QoS. An example of an aggregation of utility functions maximizes the minimum of the detection probabilities, which ensures the highest minimum quality for every video source.



(a) Source 1 **(b)** Source 2 **(c)** The aggregated QoS

Figure 4.5: Example of utility functions and QoS, which are the expected number of detected events as a function of bandwidth.

One of the most critical part of the scheduling is to define a utility function for each video source. The basic utility is the probability of detecting the next event.

Definition 4.5. The utility function, denoted by $p()$, is 0 for zero bandwidth ($p(x) = 0$). The utility function is an increasing function of the allocated resources, as

assigning more resources should not decrease the chance of detecting an event. The largest value the utility function can have is the expected number of events in the next time frame.

Figure 4.5 shows two examples of utility functions for two video sources. For Source 1 (Figure 4.5a) the probability of an event in the next period is 0.7 and no other events are possible. If we allocate less than 1 unit of bandwidth the event will be surely missed. We can crop the images to upload, requiring 1 unit of bandwidth, but this choice decreases the chance of detection. If we send the full picture in the video stream, 2 units of bandwidth is required, and we have all the possibilities to detect the event. In Source 2 (Figure 4.5b) the predictor estimates the probability of an event to 0.3. Here we can reduce the time period of sending camera pictures. If the pictures are sent through the whole time period 2 units of bandwidth should be allocated. Another option is delaying the start of transferring the camera pictures, which will result in a linear increasing utility function depending on the fraction of time the pictures are sent in the time period.

The utility is typically a multidimensional function of the resources, such as uplink bandwidth, CPU core, etc. This highlights an interesting trade-off between bandwidth and CPU. If we have more CPU but less bandwidth we may send low quality video images and run more sophisticated detection algorithms, and vice versa. Thus when the resources are scarce we are faced with the following interesting trade-off: should we use the high-performance algorithm on less data, or the low-performance one on high resolution video?

If all events in distinct video sources are naturally independent, then aggregating the utility for these events is straightforward. The simplest case is to consider the total number of detected events, and to maximize the expected number of detections. In this case the utility for each time frame is the sum of the detection probabilities given the allocated resources.

4.3.2 Dynamic programming-based solution

First let us describe the problem for n sources and B amount of available bandwidth (BW). For every source we are given a list of utility functions $p_i(x)$, corresponding to the probability of detecting an event in source i depending on the allocated bandwidth x. Our goal is to maximize the expected number of detected events, or equivalently, the aggregated detection probabilities. The problem can be formulated as a mathematical program as follows.

$$\text{maximize} \sum_{i=1}^{n} p_i(x_i), \tag{4.1}$$

$$\text{subject to} \sum_{i=1}^{n} x_i \leq B, \tag{4.2}$$

$$x_i \geq 0, \qquad i = 1, \ldots, n. \tag{4.3}$$

We will solve the mathematical program with dynamic programming, for which we need to define states. A state of the problem (c, b) consists of the source index

$$
\begin{aligned}
u_{1,0} = p_1(0) = 0, u_{1,1} = p_1(1) = 0.45,\\
u_{1,2} = p_1(2) = 0.7,\\
u_{2,0} = u_{1,0} + p_2(0) = 0,\\
u_{2,1} = \max\{u_{1,1} + p_2(0), u_{1,0} + p_2(1)\}\\
= \max\{0.45 + 0, 0 + 0.15\} = 0.45,\\
u_{2,2} = \max\{u_{1,2} + p_2(0), u_{1,1} + p_2(1),\\
u_{1,0} + p_2(2)\} = \max\{0.7, 0.575, 0.25\} = 0.7
\end{aligned}
$$

Figure 4.6: The dynamic program for $B = 2$

$c \in \{1, \ldots, n\}$ and the amount of bandwidth b already allocated to video sources $1, 2, \ldots, c$, where b is in the closed interval $[0, B]$. Note that while c needs to be an integer, as it is a video source index, there is no such restriction for b. The only restriction we have is that for every source the number of different scenarios should be finite. The state space is the set of feasible states, which is

$$\mathbf{S} = \{(c, b) : 1 \leq c \leq n, 0 \leq b \leq B\}.$$

The actual number of states depends on how many different values b can take. Let us define a working variable $u_{c,b}$ assigned to each state, which represents the QoS corresponding to sources $1, \ldots, c$ using b units of BW. The optimal $u_{c,b}$ can be computed by solving the following recursive equations. First, for a single source the QoS equals to the utility function thus we have

$$u_{1,b} = p_1(b) \ , \qquad b = 0, \ldots, B \ . \tag{4.4}$$

For the internal states optimal decisions are driven by the following equation:

$$u_{c,b} = \max_{x=0,1,\ldots,b} \{p_c(x) + u_{c-1,b-x}\}, \tag{4.5}$$

for $c = 2, \ldots, n$.

The final value we get is $u_{n,B}$, which gives us the QoS to expect. To process the states we start at source 1, and compute $u_{1,b}$ for $b = 0, \ldots B$, next compute the states for source 2 $u_{2,b}$ for $b = 0, \ldots B$, etc. Finally, we read out the the optimal x_i^* for all the sources in the opposite direction: it is the x for which $u_{i,B-b_i^*}$ takes its maximum, where $b_i^* = \sum_{j=i+1}^{n} x_j^*$.

Let us explain the dynamic program on the example of Figure 4.5. We allocate the bandwidth in integer units, and the number of sources is $n = 2$. Let the bandwidth constraint be $B = 3$. The states are $c \in \{1, 2\}$ and $b \in \{0, 1, 2, 3\}$ which is 8 states in total. See the dynamic program in Figure 4.6 till $B = 2$, and $u_{1,3} = p_1(3) = 0.7$ and the final solution is $u_{2,3} = \max\{u_{1,3} + p_2(0), u_{1,2} + p_2(1), u_{1,1} + p_2(2), u_{1,0} + p_2(3)\} = \max\{0.7+0, 0.7+0.15, 0.45+0.3, 0+0.3\} = 0.85$. See also Figure 4.5c as illustration of the QoS function.

As a straightforward generalization we consider multiple resources with a general utility function. Say we have m resources $B_1, B_2, \ldots, B_m$. Apart from bandwidth restrictions we may have further constraints on, e.g., the total number of CPU cores we may allocate for processing.

Let x_{ij} denote the amount of resource j we allocate to video source i in a given time frame. Let the set of allocations be $X = \{x_{ij}, i = 1 \ldots, n, j = 1, \ldots, m\}$. Let $p(X)$ be the gain if we make these allocations, and let $U(\cdot)$ be the desired utility.

The problem can be formulated as follows.

$$\begin{aligned} &\text{maximize } U(p(X)), \\ &\text{subject to } \sum_{i=1}^{n} x_{ij} \leq B_i, \qquad j = 1, \ldots, m, \\ &\qquad\qquad x_{ij} \geq 0, \qquad i = 1, \ldots, n, \quad j = 1, \ldots, m. \end{aligned}$$

First we have to re-define the states of the problem. A state $\mathbf{s} = (c, b_1, b_2, \ldots, b_m)$ represents the amount b_j of resource $B_j, j = 1, \ldots, m$ that has already been allocated when we consider video source c. Similarly as before, the state space is

$$\mathbf{S} = \{\mathbf{s} : 1 \leq c \leq n, 0 \leq b_i \leq B_i, i = 1, \ldots, m\}.$$

The new recursive equations that define the optimal decisions are the following. For the first source we have

$$u_{1,b_1,\ldots,b_m} = p_1(b_1, \ldots, b_m) \ , \qquad \begin{matrix} b_1=0,\ldots,B_1 \\ \vdots \\ b_m=0,\ldots,B_m \end{matrix} \ . \tag{4.6}$$

For the internal states optimal decisions are driven by the following equation:

$$u_{c,b_1,\ldots,b_m} = \max_{\substack{x_1=0,1,\ldots,b_1 \\ \vdots \\ x_m=0,1,\ldots,b_m}} \left\{ p_c(x_1, \ldots, x_m) + u_{c-1,b_1-x_1,\ldots,b_m-x_m} \right\} \tag{4.7}$$

for $c = 2, \ldots, n$.

Theorem 4.1. *The QoS can be computed in $O(nmB_{max}^{m+1})$ time, where $B_{max} = \max\{B_1, \ldots, B_m\}$.*

Proof. The maximum in (4.7) is performed over $O(B_{max}^m)$ items, and there are nmB_{max} states. □

The cloud-native application components are designed to be stateless in a sense that prediction and detection requests from the same source can be served by any active instance. This way simple load balancers available in public clouds can be used, and the system can be operated on cloud scale. The only requirement on the load balancer is to use sticky sessions to route the packets of a single video burst to the same instance. The states can be stored in the data model, in an in-memory database also provided by the cloud which is periodically loaded for the resource allocation.

4.4 Related work

Dynamic assignment of network resources has been heavily studied since the appearance of integrated communication systems [103]. Contradicting goals like service differentiation, fairness, low delay, low delay-variation, starvation avoidance have to be integrated in a properly operated network. The class of potential scheduling solutions include priority schemes with various levels of aggregations, different implementations of WFQ like resource sharing, again with various levels of aggregations. The common root of these service class based service differentiation mechanisms is the system's (or service provider's) centric optimization of resource sharing. However, in recent communication systems more short term dynamic effects are considered. The start render time of web pages is a good example for the need of user oriented dynamic resource assignment. Apart of the dynamic elements of QoS, there are random fluctuations also due to propagation changes through the air interface.

Many works have targeted the dynamic nature of wireless access sharing. The considered optimization methods include several models where the parameters are optimized by numerical investigations and a set of problems that can be attacked with general stochastic optimization tools, e.g., Markov Decision Process (MDP) [104]. Those related works that turn to distributed allocation schemes mostly apply the tool set of game theory [88], [1], but some employ other modeling techniques, e.g., portfolio theory [135].

The analytical tools developed for system oriented resource sharing are not applicable for the quantitative assessment and optimization of user oriented resource sharing. The analytic potential of the stochastic game theory approach for user-oriented dynamic behavior has been recently discovered by many researchers. A wide range of dynamic resource sharing mechanisms of wireless networks have been defined through stochastic games, here we mention only those few that we think are the most closely related to our work. The authors of [7] apply a linear program formulation to find the stationary policy for maximizing throughput given power and delay constraints. In [145] a multi-level game theoretic model is given which accounts for an evolutionary game within the set of secondary spectrum users, and for a potential game played among the providers competing for larger slices of spectrum. In contrast to these works we build an abstract model for wireless access, and apply an allocation mechanism of discrete resource units.

We see in the world of the IoT that connected devices can benefit hugely from a cloud-based back-end. For offloading processing-intensive tasks cloud-based virtualization infrastructures have emerged [128] that handle data processing, data mediation, access control and billing [86] for IoT systems. With the wide availability of WiFi and LTE networks, and with the advent of the 5G technology [93], a new IoT device class has emerged, the streaming cameras [10], e.g., remote facility management solutions can benefit of security cameras following the IoT paradigm. Separating video capture and processing with standard interfaces enables a more flexible infrastructure, where the tenants decide which streams have to be processed for what ends, and a service center provides image processing and event handling.

Another related field with similar challenges is online gaming: the industry has already started to exploit the possibilities of the cloud by moving heavy computation into data centers. The clients in such gaming scenarios are only transmitting the user input to the cloud and playing out the rendered video stream [138]. The quality of the network and the shared nature of the cloud are the factors which determine the latency and as a result the QoS of cloud gaming.

For cloud gaming, the latency threshold for proper QoS is mostly defined to be 100 ms and it is further divided into network latency (80 ms) and computation latency (20 ms) in the literature. Li et al. [67] measured the wide area latency of 4 different cloud providers from 260 points in the US and except one provider, the latency is under 80 ms to the closest data center from 80% of the measurement points. Choy et al. [24] reports 70% coverage for the same 80 ms threshold using the Amazon cloud in the US. Against long tails such frameworks, e.g., Bobtail [137], can be used that allocate multiple virtual machines during the deployment, measure the latencies and shut down the underperforming instances.

Sunderesan et al. [112] investigated multiple Internet service providers and found that the upload throughput is fairly consistent, the last mile access latency varies between 10 and 30 ms and that even the network equipment at the user can influence the service quality. As a result the quality of the network at a particular endpoint has to be investigated before the deployment of a time sensitive application.

As the cloud is shared infrastructure, other tenants may have impact on the computing performance [15]. The importance of choosing the right virtual machine flavor is investigated by Wang et al. [132] against performance fluctuations. In general, more powerful virtual machines types have more logical CPU cores, and get higher scheduling priority. While it is more expensive to operate these machines the extra price has to be paid in order to get the right performance for time sensitive applications [54].

Chapter 5

Summary of scientific results

5.1 Inter-cloud business findings

In the epicenter of the envisioned 5G ecosystem compute and network resources are allocated via an NFVIaaS market across multi-administrative domains. Customers include OTT application providers who offer services to home users, enterprise application providers who provision business services, etc. On the other side of the market, providers are compute infrastructure and network operators, given the ability of online service deployments, i.e., resource allocation and service provisioning are dynamic and flexible. This is ensured by applying virtualization and SDN techniques.

In Chapter 2 models and respective analysis are provided for B2B pricing of middleman services, and B2C pricing of cloud and network services based on Stackelberg games.

Theses 1. *In my research, I evaluated how the cloud-centered service provisioning can influence the Internet structure and which factors can motivate service providers in adjusting their mediation price on virtual services. I have formalized the model as a network formation game and derived analytical results on the equilibrium conditions of it. I also modeled price-related decisions of customers and providers in this market: for customers, I showed how they formalize their requirements, and how they select the suitable allocation from the resource offerings. For providers, I modeled how they relate to each other, and I derived how they should set their end prices, given the expected characteristics of forthcoming customer demand.*

In Theorem 2.1 I have provided the price limits that ensure *status quo* for a tiered topology dictated by Assumption 2.2.

Thesis 1.1. *[27] Let G^0 contain a number of Tier-1 nodes connected in full mesh, and a tree subgraph under each Tier-1 node in which intermediary nodes have at least k children, and all leaf nodes are at the same depth t. Furthermore, any pair of leaf nodes exchange m amount of business; intermediary nodes do not act as service sellers or buyers. Let α be the expense of an extra business link. If all providers keep their price below $\frac{\alpha}{k^{2t-3}m}$, then topology G^0 is an equilibrium.*

Under a stricter assumption on the initial topology (Assumption 2.3), I derived the Tier-1 and Tier-2 equilibrium prices in a 3-tier topology (Lemmas 2.3 and 2.4).

Thesis 1.2. *[123] Let us assume a 3-tier topology of providers, initially with no transit/peering links other than the Tier-1 full mesh. Furthermore, we assume that one transit link can be built by each Tier-2 (to a Tier-1) and Tier-3 (to a Tier-2) provider for no cost.*

In equilibrium, Tier-1 providers all set their middleman prices to $\beta_1^* = \frac{n_1(n_1-1)\alpha}{2\gamma m}$, *where* n_1 *is the number of Tier-1 providers,* γ *is the fraction of businesses reaching Tier-1 and* m *is the grand sum of business matrix* M.

In equilibrium, the Tier-2 providers' middleman price is

$$\beta_2^* = \frac{\alpha}{2\delta m}\left(n_1(n_1-1) + n_2(n_2-1)\int_{\frac{2\gamma m}{n_1(n_1-1)}}^{\infty} f(\mu)d\mu\right),$$

where δ *is the fraction of business reaching Tier-2,* n_2 *denotes the number of Tier-2 providers,* $f(\mu)$ *and* $g(\mu)$ *denote the empirical distribution of the amount of business between Tier-2 and Tier-3 provider pairs, respectively.*

Turning towards the end customers of cloud and network services, in Theorem 2.2 I proved that resource orchestration is a hard problem for them in a general topology due to the joint economic-technological requirements, e.g., capacity, latency, budget (defined in Definitions 2.3 and 2.5).

Thesis 1.3. *[120, 125] The problem of finding an eligible flow with at most* b_s *budget (or the cheapest eligible flow) is NP-hard if the network is an arbitrary graph.*

As customers face an NP-hard problem with the eligible flow selection, I reduced the Stackelberg game, where providers are leaders, customers are followers, to a stochastic game among providers as players. I showed that dynamic pricing of cloud and network resources is well-suited to the nature of the trade, and derived the equilibrium prices for various topology setups with finite and infinite capacities (Lemmas 2.6, 2.7, 2.8, 2.9, 2.10, 2.11).

Thesis 1.4. *[125] If the number of requests during the resource allocation period follows geometric distribution with parameter* q*, then equilibrium cloud prices can be derived for simple topologies in which one or two data centers are reachable by the customer directly or through a series of network providers.*

Generalizing on the findings, I relaxed the capacity constraints of service requests and derived the equilibrium prices for artificial, but more complex topologies of parallel and overlapping paths with serial network providers (depicted in Figures 2.8 and 2.9, Lemmas 2.12 and 2.13, respectively). Based on the findings, I formalized the equilibrium prices of infinite capacity cloud and network providers.

Thesis 1.5. *[125] Let* t *denote a requested path and let* $\bar{t}$ *denote the set of vertices on* t *except the service access point. The following equations provide the prices for*

all providers $x \in G$:

$$0 = \sum_{s \in \mathcal{S}} \sum_{x \in t, t \in \mathcal{T}_s} \mathbb{P}(S = s, T_s = t) \left[\left(1 - F_{B_s} \left(\sum_{y \in \bar{t}} p_y \right) \right) - f_{B_s} \left(\sum_{y \in \bar{t}} p_y \right) p_x \right], \quad (5.1)$$

where $\mathcal{T}_s$ is the set of paths from the customer s to a data center, p_x is the unit price of resource provider x, F_{B_s} and f_{B_s} are the cumulative distribution and probability density function of the budget of customer s, respectively.

Besides the expectation of the customer demand, the relative location of provider resources is of paramount importance in determining income-maximizing prices. As a general observation, I showed that those data center providers who are closer to the customer can set higher prices, and in turn the closer a network provider is to data centers, the higher its price is.

5.2 Intra-cloud management and orchestration methods

Cloud computing has grown to be the *de facto* standard to host online services by offering less expensive and simpler answers for resource and application management. Distributed infrastructure, e.g., edge and fog computing, is by all accounts the significant advance forward to boost service offerings in a cost-productive way.

In Chapter 3 I proposed cloud management techniques and methods on a wide range of resource orchestration challenges, e.g., reliability, scalability, latency-awareness, and cost effective operation.

Theses 2. *I proposed a system and several methods for ensuring high reliability and ultra-low latency with economical edge resource provisioning by an online and by an offline scheduler, which handle deployment requests in a geographically widespread cloud infrastructure at large scale. I proposed a Machine Learning-based auto-scaling method in order to address the variability of application load intensity throughout the day. I analyzed the effects of various packaging options of cloud-native applications on response latency and memory footprint.*

For scalable and economical edge cloud scheduling for latency-, and operation-critical applications, I proposed a system that provides high reliability for the applications by provisioning backup compute resources on edge nodes, which I called placeholders. Preparing for single node failures, I set the capacity of the placeholders for the maximum number of Pods on any node to fail at once. The proposed algorithms to calculate the placeholders and place the Pods are proved to be fast (Lemmas 3.1, 3.2 and 3.10)

Thesis 2.1. *[121, 44, 117]. I showed that the complexity of the online scheduling, the offline re-scheduling, and the infrastructure node clustering methods have polynomial complexity.*

Building on Lemmas 3.3, 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9 I calculated the quality of the heuristic online scheduling algorithm's output in terms of the amount of resources dedicated for fail-over, i.e., the total sum of placeholders (Theorem 3.1).

Thesis 2.2. *[117] The online scheduling solution is a 3-approximation algorithm for providing joint placement of placeholders of Pods ($HEUR \leq 3OPT$).*

Kubernetes [66], the widely used cloud manager platform, has a built-in solution for scaling its managed applications. Horizontal Pod Autoscaler (HPA) is influenced by several parameters. There are cluster level settings, e.g., down-scaling stabilization, Pod synchronization period and scaling tolerance, and there are HPA level parameters, such as minimum and maximum Pod number, scaling threshold according to arbitrary metrics. By default HPA is based on CPU utilization measurements: HPA periodically fetches monitoring data from the system, and takes a decision on how many Pods the cluster should have. In order to analyze this auto-scaling method's performance, I built lossy and lossless models to mimic HPA's behavior (Section 3.5.1).

Thesis 2.3. *[122, 126] I proposed analytical models to describe the behavior of Kubernetes' auto-scaling engine.*

Based on numerical evaluations, the models proved to be acceptable for simulation benchmarks representing the default auto-scaler's performance. Therefore I used those for comparison to evaluate my proposed ML ensemble-based auto-scaling method (Section 3.5.2). I have found it important to utilize various ML models for application usage predictions that have substantial differences in their operation. In addition to the well-known strength of ensemble-based models, the numerical results demonstrated that there was no single method that would always approximate well the optimal scaling as the input traces showed varying dynamics throughout the day, methods performed well or poorly episodically.

Thesis 2.4. *[122, 126] I proposed a Machine Learning ensemble-based predictive auto-scaling method for cloud-deployed applications. The proposed method reaches significant cost savings (up to 50%) compared to the default benchmark, mostly owing to heavily reduced request losses.*

The cost of scaling is greatly determined by the organization of application into scaling units. On one hand, the co-location of application components within the cloud results in lower operational delays, hence better QoS for the application user. On the other hand, less modularity results in superfluous resource consumption during scale-out regimes (Section 3.6). Based on Lemmas 3.12 and 3.13,

Thesis 2.5. *[118] I proposed an analytical model to describe the trade-off between resource footprint overhead during scale-out periods and the latency overhead due to organizing application code into several scaling units. Let ρ denote the resource demand and let the scaling factor be its monotone increasing function $\sigma(\rho)$. Then*

$$\frac{d\sigma}{d\rho}\rho_L + \sigma - \sigma_R = 0 \tag{5.2}$$

must hold for the scaling group borders for a minimal resource overhead, i.e., the borders of neighboring scaling groups when ordered by their scaling factors along the cumulative resource demand dimension. ρ_L denotes the width of the scaling group to the left, σ is the scaling factor value that belongs to the scaling group on the left, and σ_R denotes the scaling factor of the scaling group to the right in this arrangement.

As consequence, there are relatively large jumps between the scaling groups' scaling factors in a setting that is optimized for scaling cost.

5.3 Resource allocation of cloud access

Quality of experience in wireless access networks has been and, with the proliferation of the Networked Society, will be an important factor in telecommunications. As these networks are usually shared among multiple customers, the network resource provisioning is of paramount importance in regards the QoS the clients eventually perceive while accessing cloud applications. In Chapter 4 I therefore introduce two service quality assurance frameworks with which the existing tools in the hand of network operators are extended with the capability of *user-driven* or cloud-based quality control.

Theses 3. *I proposed an uncoordinated and a coordinated resource allocation scheme for sharing network bandwidth among cloud application clients on the same access network.*

First, I define a model for uncoordinated resource provisioning implemented as an auction where extra bandwidth for short time periods can be gained by submitting bids to the network operator which will allocate the resources to users based on their bids for the upcoming time slots (Section 4.2).

Thesis 3.1. *[124] I proposed an uncoordinated resource allocation model for accessing cloud applications based on second-price auctions and via numerical analysis I showed heuristic policies that prevail others in different traffic load scenarios.*

The results show that *generous* bidding strategy pays off on the long run when the average system load is low because in most cases the relatively low bid is enough to get jobs boosted, and the budget is saved for bursty times. When the system is close to its saturation it is worth bidding with the actual value of the jobs to-be-boosted (*rational policy*) in order to beat users applying other bidding strategies and to get jobs boosted as soon as possible.

In a centralized resource allocation scheme I proposed to ensure resource-efficiency by allowing the cloud to control the rate of data streams sent for processing to the cloud. In the illustrative video stream processing IoT system the cloud predicts the location and timing of the next possible events that need to be captured within multiple video streams. Based on the prediction some video streams are awarded more bandwidth and other cameras' feeds are switched off or instructed to operate at lower bandwidth settings. This way the shared uplink is mostly occupied with data important for the analysis (Section 4.3).

Thesis 3.2. *[119] I proposed a coordinated resource allocation scheme for accessing cloud applications that are able to predict subsequent demand for data transfer and the QoS implications of provisioned network capacity. I proposed a dynamic programming-based solution to the QoS maximization problem, and proved that optimum can be computed in $O(nmB_{max}^{m+1})$ time, where n denotes the number of data sources, m denotes the number of different types of resources to allocate, e.g., bandwidth, CPU in the cloud, $\{B_1, \ldots, B_m\}$ indicate the available amount of the respective resources, and $B_{max} = \max\{B_1, \ldots, B_m\}$.*

Note that for every source and for every type of resources, the number of different allocation scenarios should be finite. The actual number of states depends on how many different values such an allocated resource chunk can take. In case the resulting state space is relatively small, e.g., the set of feasible states is shrunk due to a coarse grained division of the total available bandwidth among the data sources, the runtime of the algorithm can be decreased.

Bibliography

[1] Khajonpong Akkarajitsakul, Ekram Hossain, and Dusit Niyato. Distributed resource allocation in wireless networks under uncertainty and application of bayesian game. *IEEE Communications Magazine*, 49(8):120–127, 2011.

[2] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Autonomic vertical elasticity of docker containers with elasticdocker. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017.

[3] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2):430–447, 2018.

[4] Muhammad Alam, Joao Rufino, Joaquim Ferreira, Syed Hassan Ahmed, Nadir Shah, and Yuanfang Chen. Orchestration of microservices for iot using docker and edge computing. *IEEE Communications Magazine*, 56(9):118–123, 2018.

[5] Ahmed Ali-Eldin, Maria Kihl, Johan Tordsson, and Erik Elmroth. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *ScienceCloud, Proceedings of the 3rd workshop on Scientific Cloud Computing*. ACM, 2012.

[6] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *2012 IEEE Network Operations and Management Symposium*, 2012.

[7] Eitan Altman, Konstantin Avratchenkov, Nicolas Bonneau, Merouane Debbah, Rachid El-Azouzi, and Daniel Sadoc Menasche. Constrained stochastic games in wireless networks. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, 2007.

[8] Amazon. Amazon EC2 spot instances, . URL `https://aws.amazon.com/ec2/spot/`.

[9] Amazon. Amazon Web Services (AWS): Elastic Container Service (ECS as CaaS) and Lambda (FaaS). `https://aws.amazon.com/`, .

[10] Aamir Nizam Ansari, Mohamed Sedky, Neelam Sharma, and Anurag Tyagi. An internet of things approach for motion detection using raspberry pi. In *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, 2015.

[11] Hamid Arabnejad, Claus Pahl, Pooyan Jamshidi, and Giovani Estrada. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.

[12] Atakan Aral, Ivona Brandic, Rafael Brundo Uriarte, Rocco De Nicola, and Vincenzo Scoca. Addressing application latency requirements through edge scheduling. *Journal of Grid Computing*, 17(12), 2019.

[13] Esteban Arcaute, Ramesh Johari, and Shie Mannor. Network formation: Bilateral contracting and myopic dynamics. *IEEE Transactions on Automatic Control*, 54(8):1765–1778, 2009.

[14] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. A discrete-time feedback controller for containerized cloud applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016.

[15] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, 2010.

[16] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[17] Dario Bega, Marco Gramaglia, Albert Banchs, Vincenzo Sciancalepore, Konstantinos Samdanis, and Xavier Costa-Perez. Optimising 5g infrastructure markets: The business of network slicing. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[18] Joseph Bertrand. Theorie mathematique de la richesse sociale. *Journal de Savants*, 67:499–508, 1883.

[19] Sumit Kumar Bose, Scott Brock, Ronald Skeoch, and Shrisha Rao. CloudSpider: Combining replication with scheduling for optimizing live migration of virtual machines across wide area networks. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011.

[20] Pablo Caballero, Albert Banchs, Gustavo de Veciana, and Xavier Costa-Pérez. Network slicing games: Enabling customization in multi-tenant networks. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[21] J.P.S. Catalão, S.J.P.S. Mariano, V.M.F. Mendes, and L.A.F.M. Ferreira. Short-term electricity prices forecasting in a competitive market: A neural network approach. *Electric Power Systems Research*, 77(10):1297–1304, 2007.

[22] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for

connection-intensive internet services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2008.

[23] Michael Chima Ogbuachi, Chinmay Gore, Anna Reale, Péter Suskovics, and Benedek Kovács. Context-Aware K8S Scheduler for Real Time Distributed 5G Edge Computing Applications. In *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2019.

[24] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, 2012.

[25] Jacomo Corbo and David Parkes. The price of selfish behavior in bilateral network formation. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing*, 2005.

[26] Costas Courcoubetis. *Pricing Communication Networks Economics, Technology and Modelling*. Wiley Online Library, 2003.

[27] Mate Cserep, Akos Recse, Robert Szabo, and Laszlo Toka. Business network formation among 5G providers. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[28] Amogh Dhamdhere and Constantine Dovrolis. The internet is flat: Modeling the transition from a transit hierarchy to a peering mesh. In *Proceedings of the 6th International COnference*, Co-NEXT '10. ACM, 2010.

[29] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow. In *ICAS 2011 : The Seventh International Conference on Autonomic and Autonomous Systems*, 2011.

[30] Ericsson. Network slicing. `https://www.ericsson.com/en/network-slicing`.

[31] ETSI. White Paper: Network Functions Virtualisation (NFV). `http://portal.etsi.org/nfv/nfv_white_paper2.pdf`, 2013.

[32] ETSI. Management and Orchestration. Technical report, ETSI GS NFV-MAN 001, 12 2014.

[33] ETSI. Network Service Templates Specification. Technical report, ETSI GS NFV-IFA 014, 10 2016.

[34] Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing*, 2003.

[35] Jingyuan Fan, Meiling Jiang, Ori Rottenstreich, Yangming Zhao, Tong Guan, Ram Ramesh, Sanjukta Das, and Chunming Qiao. A framework for provisioning availability of nfv in data center networks. *IEEE Journal on Selected Areas in Communications*, 36(10):2246–2259, 2018.

[36] I. Farris, T. Taleb, H. Flinck, and A. Iera. Providing ultra-short latency to user-centric 5g applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies*, 29(4), 2018.

[37] Fortio. Fortio. `http://fortio.org/`.

[38] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[39] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management*, 2010.

[40] Google. Google Cloud: Google Kubernetes Engine (GKE as Caas) and Google Cloud Functions (FaaS). `https://cloud.google.com/`.

[41] Akhil Gupta and Rakesh Kumar Jha. A survey of 5G network: Architecture and emerging technologies. *IEEE Access*, 3:1206–1232, 2015.

[42] David M. Gutierrez-Estevez, Marco Gramaglia, Antonio De Domenico, Ghina Dandachi, Sina Khatibi, Dimitris Tsolkas, Irina Balan, Andres Garcia-Saavedra, Uri Elzur, and Yue Wang. Artificial intelligence for elastic management and orchestration of 5g networks. *IEEE Wireless Communications*, 26(5):134–141, 2019.

[43] David Haja, Marton Szabo, Mark Szalay, Adam Nagy, Andras Kern, Laszlo Toka, and Balazs Sonkoly. How to orchestrate a distributed OpenStack. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[44] David Haja, Mark Szalay, Balazs Sonkoly, Gergely Pongracz, and Laszlo Toka. Sharpening Kubernetes for the Edge. In *ACM SIGCOMM Conference Posters and Demos*, 2019.

[45] David Haja, Zoltan Richard Turanyi, and Laszlo Toka. Location, Proximity, Affinity – The key factors in FaaS. *Infocommunications Journal*, 12(4):14–21, 2020.

[46] J. Halpern and C. Pignataro. Service Function Chaining (SFC) Architecture. IETF RFC 7665, 2015.

[47] Jeff Hawkins and Sandra Blakeslee. *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan, 2007.

[48] Yan he Jia, Li xin Tang, Zhe George Zhang, and Xiao feng Chen. MMPP/M/C queue with congestion-based staffing policy and applications in operations of steel industry. *Springer Journal of Iron and Steel Research International*, 26 (7):659—-668, 2018.

[49] Poul E. Heegaard, Gergely Biczok, and Laszlo Toka. Sharing is power: Incentives for information exchange in multi-operator service delivery. In *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016.

[50] Cheol-Ho Hong and Blesson Varghese. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *ACM Computing Surveys*, 52(5), 2019.

[51] Shay Horovitz and Yair Arian. Efficient Cloud Auto-Scaling with SLA Objective Using Q-Learning. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2018.

[52] HPA. Horizontal Pod Autoscaler - Kubernetes. `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`.

[53] Eduardo Huedo, Rubén S. Montero, Rafael Moreno-Vozmediano, Constantino Vázquez, Vlastimil Holer, and Ignacio M. Llorente. Opportunistic deployment of distributed edge clouds for latency-critical applications. *Journal of Grid Computing*, 19(1), 2021.

[54] Alexandru Iosup, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011.

[55] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

[56] ISP. ISP 3-Tier Model. Available online: `https://www.thousandeyes.com/learning/techtorials/isp-tiers`.

[57] Nidhi Jain and Inderveer Chana. Energy-aware Virtual Machine Migration for Cloud Computing - A Firefly Optimization Approach. *Journal of Grid Computing*, 14(6), 2016.

[58] Asad Javed, Jérémy Robert, Keijo Heljanko, and Kary Främling. IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications. *Journal of Grid Computing*, 18(3), 2020.

[59] Mak A. Kaboudan. A dynamic-server queuing simulation. *Computers & Operations Research*, 25(6):431–439, 1998.

[60] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th International conference on Autonomic computing*, 2009.

[61] Yossi Kanizo, Ori Rottenstreich, Itai Segall, and Jose Yallouz. Optimizing virtual backup allocation for middleboxes. *IEEE/ACM Transactions on Networking*, 25(5):2759–2772, 2017.

[62] Hamzeh Khazaei, Rajsimman Ravichandiran, Byungchul Park, Hadi Bannazadeh, Ali Tizghadam, and Alberto Leon-Garcia. Elascale: Autoscaling and monitoring as a service. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, 2017.

[63] Selma Khebbache, Makhlouf Hadji, and Djamal Zeghlache. Scalable and cost-efficient algorithms for vnf chaining and placement problem. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, 2017.

[64] P. S. Khodashenas, C. Ruiz, J. Ferrer Riera, J. O. Fajardo, I. Taboada, B. Blanco, F. Liberal, J. G. Lloreda, J. Pérez-Romero, O. Sallent, I. Neokosmidis, and T. Rokkas. Service provisioning and pricing methods in a multi-tenant cloud enabled ran. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2016.

[65] Joanna Kosińska and Krzysztof Zielinski. Autonomic management framework for cloud-native applications. *Journal of Grid Computing*, 18, 12 2020.

[66] Kubernetes. Kubernetes: Production-Grade Container Orchestration. `https://kubernetes.io`.

[67] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010.

[68] Shuhui Li, D.C. Wunsch, E.A. O'Hair, and M.G. Giesselmann. Using neural networks to estimate wind turbine power generation. *IEEE Transactions on Energy Conversion*, 16(3):276–282, 2001.

[69] Aemen Lodhi, Amogh Dhamdhere, and Constantine Dovrolis. GENESIS: An agent-based model of interdomain network formation, traffic flow and economics. In *IEEE INFOCOM*, 2012.

[70] Aemen Lodhi, Amogh Dhamdhere, and Constantine Dovrolis. Open peering by internet transit providers: Peer preference or peer pressure? In *IEEE INFOCOM*, 2014.

[71] Tania Lorido-Botrán, Jose Miguel-Alonso, and Jose Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592, 2014.

[72] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *ACM Symposium on Cloud Computing (SoCC'21)*, 2021.

[73] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Jian He, and Cheng-Zhong Xu. An in-depth study of microservice call graph and runtime performance. *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[74] Nguyen Cong Luong, Ping Wang, Dusit Niyato, Yonggang Wen, and Zhu Han. Resource management in cloud networking using economic analysis and pricing models: A survey. *IEEE Communications Surveys & Tutorials*, 19(2): 954–1001, 2017.

[75] Pavel Mach and Zdenek Becvar. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[76] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[77] Vladimir Mazalov and Andrei Gurtov. Queuing system with on-demand number of servers. *Mathematica Applicanda*, 40(2):1–12, 2012.

[78] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[79] Valter Rogério Messias, Julio Cezar Estrella, Ricardo Ehlers, Marcos José Santana, Regina Carlucci Santana, and Stephan Reiff-Marganiec. Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. *Springer Neural Computing & Applications*, 27(8):2383–2406, 2016.

[80] Microsoft Azure. Microsoft Azure: Azure Kubernetes Service (AKS as CaaS) and Azure Functions (FaaS). `https://azure.microsoft.com/`.

[81] Jeremy Miles. R squared, adjusted R squared. *Wiley StatsRef: Statistics Reference Online*, 2014.

[82] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2018.

[83] Mithun Mukherjee, Lei Shu, and Di Wang. Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges. *IEEE Communications Surveys & Tutorials*, 20(3):1826–1857, 2018.

[84] Kento Nakanishi, Fumiya Suzuki, Satoshi Ohzahata, Ryo Yamamoto, and Toshihiko Kato. A container-based content delivery method for edge cloud over wide area network. In *2020 International Conference on Information Networking (ICOIN)*, 2020.

[85] NASA. NASA-HTTP logs. ftp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

[86] Stefan Nastic, Sanjin Sehic, Duc-Hung Le, Hong-Linh Truong, and Schahram Dustdar. Provisioning software-defined iot cloud systems. In *2014 International Conference on Future Internet of Things and Cloud*, 2014.

[87] Bram Naudts, Mario Flores, Rashid Mijumbi, Sofie Verbrugge, Joan Serrat, and Didier Colle. A dynamic pricing algorithm for a network of virtual resources. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016.

[88] Dusit Niyato and Ekram Hossain. Competitive pricing for spectrum sharing in cognitive radio networks: Dynamic game, inefficiency of nash equilibrium, and collusion. *IEEE Journal on Selected Areas in Communications*, 26(1): 192–202, 2008.

[89] William B. Norton. Internet service providers and peering. In *NANOG*, volume 19, pages 1–17, 2001.

[90] Balázs Németh, Márk Szalay, János Dóka, Matthias Rost, Stefan Schmid, László Toka, and Balázs Sonkoly. Fast and efficient network service embedding method with adaptive offloading to the edge. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[91] ONF. Software-defined networking: The new norm for networks (white paper), 2012. https://www.opennetworking.org.

[92] Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, 1994.

[93] Maria Rita Palattella, Mischa Dohler, Alfredo Grieco, Gianluca Rizzo, Johan Torsner, Thomas Engel, and Latif Ladid. Internet of Things in the 5G Era: Enablers, Architecture, and Business Models. *IEEE Journal on Selected Areas in Communications*, 34(3):510–527, 2016.

[94] Nisha Panwar, Shantanu Sharma, and Awadhesh Kumar Singh. A survey on 5g: The next generation of mobile communication. *Physical Communication*, 18(P2):64–84, 2016.

[95] A.K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[96] Prometheus. Prometheus. https://prometheus.io/.

[97] M. Reza Rahimi, Jian Ren, Chi Harold Liu, Athanasios V. Vasilakos, and Nalini Venkatasubramanian. Mobile cloud computing: A survey, state of art and future directions. *Mobile Networks and Applications*, 19(2):133–143, 2014.

[98] Joy Rahman and Palden Lama. Predicting the end-to-end tail latency of containerized microservices in the cloud. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, 2019.

[99] Ali Rajabi and Johnny W. Wong. MMPP characterization of web application traffic. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2012.

[100] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Le Wang, and Gang Yin. VCONF: A reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th International Conference on Autonomic Computing, ICAC'09*, 2009.

[101] Gourav Rattihalli, Madhusudhan Govindaraju, Hui Lu, and Devesh Tiwari. Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019.

[102] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. *ACM Computing Surveys*, 52(6), 2019.

[103] James Roberts, Ugo Mocci, and Jorma Virtamo. *Broadband Network Teletraffic*. Springer, 1996.

[104] Sheldon M. Ross. *Applied Probability Models with Optimization Applications*. Holden-Day, San Francisco, 1970.

[105] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019.

[106] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. Autopilot: Workload autoscaling at google. In *Proceedings of the Fifteenth European Conference on Computer Systems*. ACM, 2020.

[107] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019.

[108] Srinivas Shakkottai and Rayadurgam Srikant. Economics of network pricing with multiple ISPs. *IEEE/ACM Transactions on Networking*, 14(6):1233–1245, Dec 2006. ISSN 1063-6692. DOI: 10.1109/TNET.2006.886393.

[109] Xiaojun Shang, Yaodong Huang, Zhenhua Liu, and Yuanyuan Yang. Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme. In *IEEE Conference on Computer Communications (INFOCOM)*, 2020.

[110] Balázs Sonkoly, Dávid Haja, Balázs Németh, Márk Szalay, János Czentye, Róbert Szabó, Rehmat Ullah, Byung-Seo Kim, and László Toka. Scalable

edge cloud platforms for iot services. *Journal of Network and Computer Applications*, 170:102785, 2020.

[111] Balázs Sonkoly, Róbert Szabó, Balázs Németh, János Czentye, Dávid Haja, Márk Szalay, János Dóka, Balázs P. Gerő, Dávid Jocha, and László Toka. 5G applications from vision to reality: Multi-operator orchestration. *IEEE Journal on Selected Areas in Communications*, 38(7):1401–1416, 2020.

[112] Srikanth Sundaresan, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: A view from the gateway. In *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011.

[113] Mark Szalay, Peter Matray, and Laszlo Toka. Minimizing state access delay for cloud-native network functions. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, 2019.

[114] Mark Szalay, Maté Nagy, Daniel Gehberger, Zoltan Kiss, Peter Matray, Felician Nemeth, Gergely Pongracz, Gabor Retvari, and Laszlo Toka. Industrial-Scale Stateless Network Functions. In *2019 IEEE International Conference on Cloud Computing (CLOUD)*, 2019.

[115] Mark Szalay, Peter Matray, and Laszlo Toka. State management for cloud-native applications. *Electronics*, 10(4), 2021.

[116] Eva Tardos and Tom Wexler. *Network Formation Games and the Potential Function Method*, page 487–516. Cambridge University Press, 2007.

[117] Laszlo Toka. Ultra-reliable and low-latency computing in the edge with Kubernetes. *Journal of Grid Computing*, 19(3), 2021.

[118] Laszlo Toka. The shape of your cloud: How to design and run polylithic cloud applications. *IEEE Access*, 10:97971–97982, 2022.

[119] Laszlo Toka, Balazs Lajtha, Eva Hosszu, Bence Formanek, Daniel Gehberger, and Janos Tapolcai. A resource-aware and time-critical IoT framework. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[120] Laszlo Toka, Janos Tapolcai, George Darzanos, and Balazs Sonkoly. On pricing of 5G services. In *IEEE Global Communications Conference (GLOBECOM)*, 2017.

[121] Laszlo Toka, David Haja, Attila Korosi, and Balazs Sonkoly. Resource provisioning for highly reliable and ultra-responsive edge applications. In *IEEE International Conference on Cloud Networking (ClOUDNET)*, 2019.

[122] Laszlo Toka, Gergely Dobreff, Balazs Fodor, and Balazs Sonkoly. Adaptive AI-based auto-scaling for Kubernetes. In *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020.

[123] Laszlo Toka, Akos Recse, Mate Cserep, and Robert Szabo. On the mediation price war of 5G providers. *Electronics*, 9(11), 2020.

[124] Laszlo Toka, Mark Szalay, David Haja, Geza Szabo, Sandor Racz, and Miklos Telek. To boost or not to boost: a stochastic game in wireless access networks. In *IEEE International Conference on Communications (ICC)*, 2020.

[125] Laszlo Toka, Marton Zubor, Attila Korosi, George Darzanos, Ori Rottenstreich, and Balazs Sonkoly. Pricing games of NFV infrastructure providers. *Telecommunication Systems*, 76:219–232, 2020.

[126] Laszlo Toka, Gergely Dobreff, Balazs Fodor, and Balazs Sonkoly. Machine learning-based scaling management for Kubernetes edge clusters. *IEEE Transactions on Network and Service Management*, 18(1):958–972, 2021.

[127] Thinh Duy Tran and Long Bao Le. Resource allocation for multi-tenant network slicing: A multi-leader multi-follower stackelberg game approach. *IEEE Transactions on Vehicular Technology*, 69(8):8886–8899, 2020.

[128] Hong-Linh Truong and Schahram Dustdar. Principles for engineering IoT cloud systems. *IEEE Cloud Computing*, 2(2):68–76, 2015.

[129] I. Vaishnavi, J. Czentye, M. Gharbaoui, G. Giuliani, D. Haja, J. Harmatos, D. Jocha, J. Kim, B. Martini, J. MeMn, P. Monti, B. Nemeth, W. Y. Poe, A. Ramos, A. Sgambelluria, B. Sonkoly, L. Toka, F. Tusa, C. J. Bernardos, and R. Szabo. Realizing services and slices across multiple operator domains. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018.

[130] Muhammad Wajahat, Anshul Gandhi, Alexei Karve, and Andrzej Kochut. Using machine learning for black-box autoscaling. In *2016 Seventh International Green and Sustainable Computing Conference (IGSC)*, 2016.

[131] Jiafu Wan, Shenglong Tang, Zhaogang Shu, Di Li, Shiyong Wang, Muhammad Imran, and Athanasios V. Vasilakos. Software-defined industrial internet of things in the context of industry 4.0. *IEEE Sensors Journal*, 16(20):7373–7380, 2016.

[132] Guohui Wang and T. S. Eugene Ng. The impact of virtualization on network performance of Amazon EC2 data center. In *IEEE INFOCOM*, 2010.

[133] Yonggang Wen, Weiwen Zhang, and Haiyun Luo. Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. In *IEEE INFOCOM*, 2012.

[134] Worldcup. WorldCup98 HTTP logs. `ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html`.

[135] Tadeusz Wysocki and Abbas Jamalipour. Spectrum management in cognitive radio: Applications of portfolio theory in wireless communications. *IEEE Wireless Communications*, 18(4):52–60, 2011.

[136] Hong Xu and Baochun Li. A study of pricing for cloud resources. *SIGMETRICS Perform. Eval. Rev.*, 40(4):3–12, 2013.

[137] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. Bobtail: Avoiding long tails in the cloud. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013.

[138] Zheng Xue, Di Wu, Jian He, Xiaojun Hei, and Yong Liu. Playing high-end video games in the cloud: A measurement study. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2013–2025, 2015.

[139] Louiza Yala, Pantelis A Frangoudis, and Adlen Ksentini. Latency and availability driven VNF placement in a MEC-NFV environment. In *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[140] Jingjing Yao and Nirwan Ansari. Reliability-Aware Fog Resource Provisioning for Deadline-Driven IoT Services. In *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[141] Faqir Zarrar Yousaf and Tarik Taleb. Fine-grained resource-aware virtual network function management for 5G carrier cloud. *IEEE Network*, 30(2): 110–115, 2016.

[142] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Elsevier Journal of Systems Architecture*, 98:289–330, 2019.

[143] Guangba Yu, Pengfei Chen, and Zibin Zheng. Microscaler: Automatic scaling for microservices with an online learning approach. In *2019 IEEE International Conference on Web Services (ICWS)*, 2019.

[144] Jiao Zhang, Zenan Wang, Chunyi Peng, Linquan Zhang, Tao Huang, and Yunjie Liu. Raba: Resource-aware backup allocation for a chain of virtual network functions. In *IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[145] Kun Zhu, Dusit Niyato, and Ping Wang. Dynamic bandwidth allocation under uncertainty in cognitive radio networks. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, 2011.

[146] Qian Zhu and Gagan Agrawal. Resource provisioning with budget constraints for adaptive applications in cloud environments. *IEEE Transactions on Services Computing*, 5(4):497–511, 2012.

Glossary

AR Autoregressive. 60–63, 75

AS Autonomous System. 31

CPU Central Processing Unit. 53–57, 73, 74, 84, 86, 88, 90, 94, 96

HPA Horizontal Pod Autoscaler. 54–60, 62–65, 73, 94

IaaS Infrastructure as a Service. 6

IoT Internet of Things. 4, 70, 77, 83–85, 89, 95

ISP Internet Service Provider. 2, 3, 31

KPI Key Performance Indicator. 79, 80

MDP Markov Decision Process. 79, 89

ML Machine Learning. 2, 34, 52, 54, 60, 62–64, 74, 75, 93, 94

MMPP Markov-Modulated Poisson Process. 55–60, 63, 73

MSE Mean Squared Error. 58–61

NFV Network Function Virtualization. 3, 5, 6, 29

NFVIaaS Network Function Virtualization Infrastructure as a Service. 6, 29, 91

NN Neural Network. 60, 61, 74, 75

OTT Over-the-Top. 6, 91

QoS Quality of Service. 2, 4, 5, 8, 31, 33, 34, 65, 73, 78–80, 84, 85, 87–90, 94–96

RL Reinforcement Learning. 60–63, 73, 74

SaaS Software as a Service. 6

SAP Service Access Points. 7, 8, 15, 16, 20, 23, 25, 26, 28

SDN Software Defined Networking. 3, 91

SFC Service Function Chain. 5, 6, 29, 72

SLA Service Level Agreement. 4, 33, 52, 73–75

VM Virtual Machine. 2–4, 71, 72, 74, 75

VNF Virtual Network Function. 5, 6, 8, 72

VNFaaS Virtual Network Function as a Service. 6

WFQ Weighted Fair Queuing. 78, 89