# Automated Provisioning of Fairly Priced Resources

Dissertation

zur Erlangung des Doktorgrades
Ph. D.
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

im PhD Programme in Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Abhinandan Sridhara Rao Prasad
aus Mysore, Karnataka, India

Göttingen
im Juni 2018

**Betreuungsausschuss:**

Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Dr. Volker Hilt,
Nokia Bell Labs, Stuttgart

PD Dr. rer. nat. habil. Mayutan Arumaithurai,
Georg-August-Universität Göttingen

Dr. David Koll,
Georg-August-Universität Göttingen

**Prüfungskommission:**

Referent:

Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Korreferenten:

Prof. Dr. Nils Aschenbruck,
Universität Osnabrück, Germany

Weitere Mitglieder
der Prüfungskommission:

PD Dr. rer. nat. habil. Mayutan Arumaithurai,
Georg-August-Universität Göttingen
Prof. Dr. Carsten Damm,
Georg-August-Universität Göttingen
Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen
Prof. Dr. Stephen Waack,
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 21. Juni 2018

## Abstract

In recent times, cloud service providers are increasingly offering more complex services. These complex services are heterogeneous and composed *dynamically* from traditional services such as *Infrastructure as a Service* (IaaS), *Software as a Service* (SaaS), and *Platform as a Service* (PaaS) to handle ad-hoc demands. Consequently, the current cloud is a complicated *marketplace*. Furthermore, resource prices influence the user demands, and user demands eventually drive resource provisioning. Correspondingly, resource *pricing* and *provisioning* are indispensable to each other. Hence, cloud service providers face the challenge of optimizing resource prices and provisioning.

This challenge has attracted both industry and academia. However, most of the pricing approaches proposed and practiced achieve either *efficiency* or *fairness*. Thus, current pricing schemes reward either the service provider or the user. In resource provisioning, both industry and academia focus on addressing the issue of *when* to provision, while disregarding *what* to provision. Consequently, services are deployed using a single *Virtual Machine* (VM) size for all components resulting in performance degradation and eventually leading to *Service Level Objective* (SLO) violations.

This dissertation proposes *Automated Resource Pricing and Provisioning* (ARPP) for the pricing and provisioning of cloud and edge resources to address the above-mentioned challenges. The `ARPP` pricing supports three pricing approaches dubbed *Robust Auction for Edge Resource Allocation* (RAERA), *Edge Resource Market* (ERM), and *Online Fisher Market* (OFM). `RAERA` is a robust optimization-based sealed auction proposed to address *price uncertainty*. `ERM` computes *differential prices* for buyers with *Separable Piecewise-Linear Concave* (SPLC) utilities. `OFM` is an *online* Fisher market for pricing varying resources for varying buyers. Both `ERM` and `OFM` maximize *Nash Social Welfare* (NSW) – a Pareto outcome between *efficiency* and *fairness*. The evaluation demonstrates the effectiveness and scalability of proposed approaches.

In this dissertation, we propose *Robust Configuration* (RConf) and *Robust Configuration Primal-Dual* (RConfPD) as an answer to automate the issue of *what* to provision. `RConf` finds an *optimal* configuration for maximizing the overall resource utilization of a complex service. Conversely, `RConfPD` trades off resource utilization for performance. Hence, it is appropriate for services with nearly instant provisioning requirements. Both the approaches estimate performance cost for arbitrary arrivals and departures using a robust queueing theory-based model. The experimental evaluations show the overall resource utilization

improvement of $16-50\%$ over one-size-fits-all solutions, and simultaneously deploys $22\%$ of fewer resources.

The ARPP proposed in this dissertation can be integrated with edge or cloud orchestrators. Moreover, the ideas presented can also be applied to (i) pricing and provisioning *Network Function Virtualization* (NFV) service chains, (ii) building a single *marketplace* for cloud, edge, and fog resources and applying ARPP ERM to price resources differentially according to resource types, and (iii) automatically selecting complex service type (cloud, edge or fog) and provisioning depending on user preferences.

**Acknowledgements**

It is my pleasure to thank those who made this dissertation possible. I would like to sincerely thank my supervisor Prof. Dr. Xiaoming Fu for his constant support to pursue my diverse research interests.

I gratefully acknowledge the funding received towards my Ph.D. from the EU FP7 Marie Curie Actions through CleanSky ITN project.

I am sincerely thankful to Dr. Volker Hilt, who also kindly agreed to be my second dissertation supervisor. Volker provided me an opportunity to pursue research in Bell Labs at Stuttgart and Dublin.

I am grateful to my former mentor Dr. David Koll, who also kindly agreed to be my third dissertation superior. David frequently provided constructive criticism and reviews in hours over hours of discussions in dozens of meetings, which significantly contributed to the quality of dissertation.

I am obliged to my colleague Dr. Mayutan Arumaithurai for his constant encouragement and invaluable suggestions, which have significantly improved my work.

I am grateful to Dr. Jesus Omana Iglesias and Dr. Jordi Arjona Aroca for hosting me in Bell Labs, Ireland. Their constructive criticisms and reviews have helped me to significantly improved my work.

I am obliged to Prof. Carsten Damm, Prof. Dieter Hogrefe, Prof. Nils Aschenbruck and Prof. Stephen Waack for being members of my examination committee, and to Prof. Yuming Jiang, for hosting me during the research visit to NTNU, Trondheim.

I sincerely thank my former and current colleagues at the Computer Networks Group at the University of Göttingen, especially Sameer Kulkarni, Osamah Barakat, Sripriya Srikant Adhatarao and Jacopo De Benedetto whose feedback also contributed to the quality of this dissertation. Furthermore, I thank Annette Kadziora, Federica Poltronieri, and Tina Bockler for taking care of all the administrative procedures.

I am deeply obliged to Dr. P. Suresh and Prof. Shrisha Rao (International Institute of Information Technology), for inspiring me to pursue research.

Last and most importantly, I am indebted to my family: to my parents S. Prasad and S. Usha Prasad and my wife Ashwini Abhinandan, for their unrelenting support during difficult times, and for their exceptional personal sacrifices to enable my success.

The research leading to these results has received funding from the EU FP7 Marie Curie Actions by the EC Seventh Framework Programme (FP7/2007-2013) Grant Agreement No. 607584 (the Cleansky project). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the CleanSky project or the European Commission.

# Contents

# List of Figures

# List of Tables

# List of Definitions, Lemma and Theorems

# Acronyms

**AR** *Auto Regressive*

**ARIMA** *Auto Regressive Integrated and Moving Average*

**ARPP** *Automated Resource Pricing and Provisioning*

**ARU** *Average Resource Utilization*

**AWS** *Amazon Web Services*

**CDF** *Cumulative Distribution Function*

**CES** *Constant Elasticity of Substitution*

**CPU** *Central Processing Unit*

**DB** *Database*

**DC** *Datacenter*

**DFS** *Depth First Search*

**DPI** *Deep Packet Inspection*

**ERM** *Edge Resource Market*

**EC2** *Elastic Compute Cloud*

**FW** *Firewall*

**GB** *Gigabyte*

**IaaS** *Infrastructure as a Service*

**IC** *Incentive Compatible*

**ILP** *Integer Linear Program*

**IoT** *Internet of Things*

**IR** *Individual Rationality*

**ISP** *Internet Service Provider*

**KKT** *Karush-Kuhn-Tucker*

**KPI** *Key Performance Indicator*

**LB** *Load Balancer*

**MA** *Moving Average*

**MAE** *Mean Absolute Error*

**MMKP** *Multiple choice Multidimensional Knapsack Problem*

**NAT** *Network Address Translation*

**NFV** *Network Function Virtualization*

**NIST** *National Institute of Standards and Technology*

**NSW** *Nash Social Welfare*

**OCO** *Online Convex Optimization*

**OFM** *Online Fisher Market*

**OMD** *Online Mirror Descent*

**PaaS** *Platform as a Service*

**PTAS** *Polynomial-Time Approximation Scheme*

**QoE** *Quality of Experience*

**QoS** *Quality of Service*

**RAERA** *Robust Auction for Edge Resource Allocation*

**RAM** *Random Access Memory*

**RConf** *Robust Configuration*

**RConfPD** *Robust Configuration Primal-Dual*

**RO** *Robust Optimization*

**SaaS** *Software as a Service*

**SFC** *Service Function Chains*

**SLO** *Service Level Objective*

**SPLC** *Separable Piecewise-Linear Concave*

**VCP** *Virtual Configuration Problem*

**VM** *Virtual Machine*

**VNF** *Virtual Network Function*

**WS** *Webserver*

# Chapter 1

# Introduction

*It's faster in every case to talk to the server than it is my local hard disk... Carrying around these non-connected computers – with tons of data and state in them – is byzantine by comparison.*
*You've got to start with the customer experience and work backwards to the technology.*

— Steve Jobs, Worldwide Developer Conference, May 1997 [1]

## 1.1 Complex Services

The *National Institute of Standards and Technology* (NIST) defines cloud computing as [2]: "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*".

In essence, cloud computing is the paradigm of offering computing services over the internet. These services can scale on demand to support ad-hoc workload. Hence, cloud computing is one of the primary enablers of new paradigms, such as edge computing and Network Function Virtualization (NFV). Earlier the cloud offerings were limited to infrastructure (e.g., virtual machines), development environments and applications. These offerings are popularly known as *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS) respectively. Recent improvements in virtualization and customer requirements are driving cloud service providers to offer *complex services* deployed on *virtualized* resources [3]. These resources could vary from physical infrastructures, such as servers and software components and are hosted in data centers. In cloud computing, a chain consisting of a load balancer, web server and database as illustrated in Figure 1.1 is a simple example of a complex service. Similarly, *Service Function Chains* (SFC) in computer networking is a complex service. For instance, an SFC consisting of *Network Address Translation* (NAT), firewall, and *Deep Packet Inspection* (DPI) as depicted in Figure 1.2 is an example of a complex service.

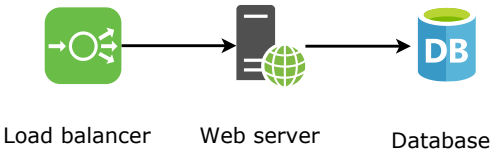



Figure 1.1: A complex service consisting of a load balancer, web server, and database

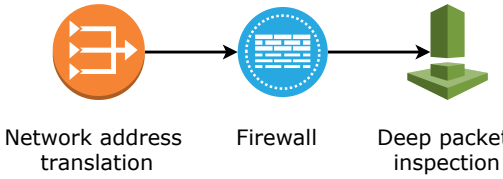



Figure 1.2: A service function chain consisting of a NAT, firewall, and DPI

Without loss of generality, we can define *complex service* as follows:

**Definition 1.1 (Complex Service)** A complex service is a chain of virtual components that together process a certain fraction of the traffic passing through the provider's network.

Figure 1.3 illustrates our complex service model. The model is generic and abstracts the behavior of existing complex services such as NFV service chains. In a service chain, each component processes the incoming traffic flows before it leaves the service chain as depicted in Figure 1.3. In this dissertation, we focus mainly on the complex service model with virtualized components (simply put, they are running on virtual instances such as *Virtual Machines* (VMs) and containers).

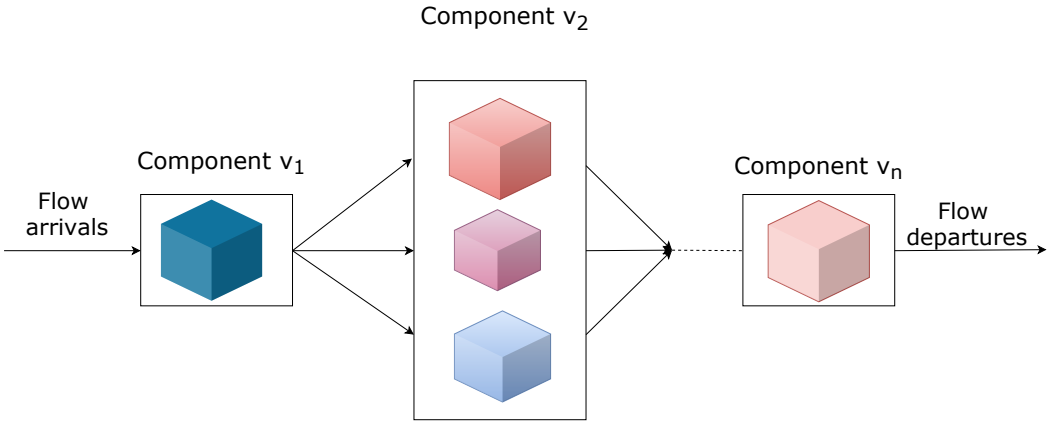



Figure 1.3: The abstract model of a complex service consisting of many components, each of which may consist of several instances. Each component processes a fraction of the traffic.

With the advent of edge computing, cloud service providers are increasingly moving data centers closer to the network edge (i.e., the customer) to provide better customer experience. This approach not only reduces latency but also improves the scalability of complex services [4–7]. In edge, users arrive at the service provider for the complex services such as virtual networks and VMs. Based on the customer location and preferences, the service provider selects appropriate edge resources and offers to the user. Figure 1.4 illustrates the typical edge computing scenario where cloud service providers provide complex services to the customers.

For the remainder of this dissertation, the term *service* is a short form of *complex services*. Similarly, the term *service providers* are a short form of *cloud service providers*.

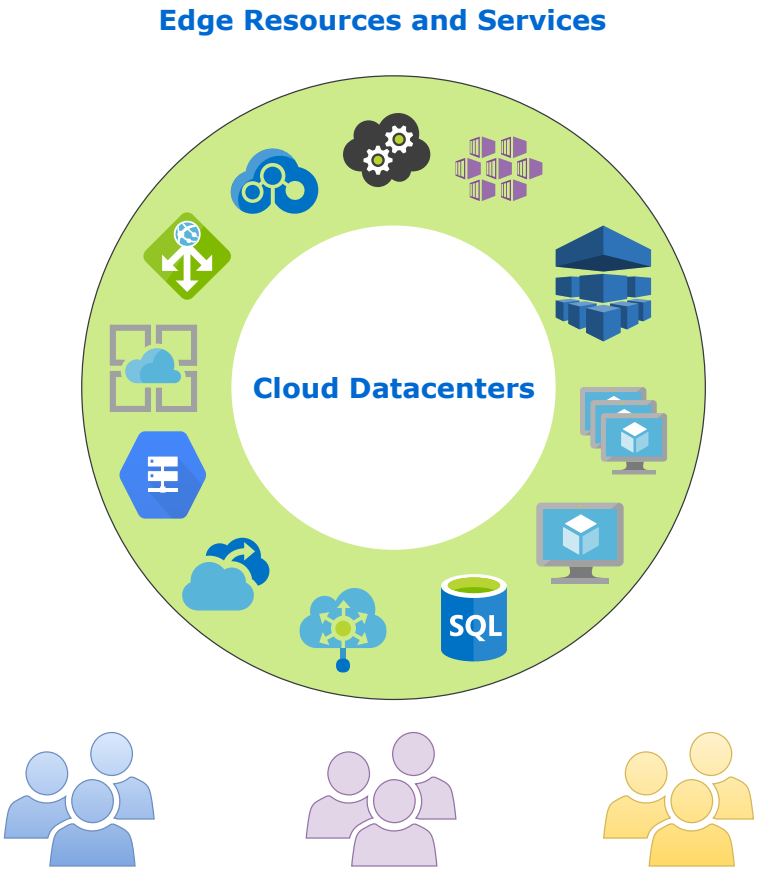



Figure 1.4: Complex services, resources, and users in edge computing.

## 1.2 The Problem

Currently, many cloud service providers are offering complex services on top of traditional cloud services, such as IaaS, SaaS, and PaaS. These services are heterogeneous and charged differently. For example, SaaS is billed per user or on application basis while IaaS such as VM is billed hourly. Most often, IaaS or PaaS act as the backend for SaaS and complex services as illustrated in Figure 1.3. Any IaaS outrage impacts SaaS, for instance, inter and intra *Datacenter* (DC) problems affected around 32% of SaaS applications [8]. Hence, the performance of SaaS and complex services depend predominantly on IaaS or PaaS.

Cloud users are the customers for SaaS providers, in turn, the customers of IaaS or PaaS providers. This relationship among cloud users, SaaS, IaaS, and PaaS providers result in a complicated marketplace affecting both resource allocation and performance of the service [9]. Moreover, the service hosting jointly depends on pricing, advertised and delivered SLOs. The prices and advertised SLOs influence the user demands and user demands eventually drive the resource provisioning [10]. It is well-known that *resource provisioning* is the allocation of resources to satisfy SLO for an incoming workload [11]. Conversely, resource provisioning is responsible for delivering SLOs which in turn affect user demand and eventually affect prices. Figures 1.5 and 1.6 depict the relationship between resource pricing and provisioning. Hence, both resource pricing and provisioning are interdependent. Hence, any optimization on pricing should consider resource provisioning and vice versa. Therefore, in this dissertation, we present solutions to automatically price and provision complex services in both edge and cloud environments. This process involves two significant steps namely i) resource pricing and ii) provisioning.

### 1.2.1 Pricing

*Pay-as-you-go* or *Fixed pricing* is the current pricing scheme followed by most of the cloud service providers. In fixed pricing, consumers are charged based on resource usage. For instance, Amazon EC2 instances are billed on an hourly basis. In fixed pricing, revenue is maximized only if every customer behavior is *well-defined* and arrivals are *temporally invariant* [12]. For instance, consumers spending more money on weekends than weekdays are an example of a well-defined behavior. The temporal invariance implies constant customer arrivals. However, these conditions are not valid in cloud computing because both customer demands and arrivals are ad-hoc [13]. Moreover, the physical capacity of cloud resources is finite [14]. Most often, the default fixed pricing favors cloud service providers contractually [15]. Further, current cloud resource prices are *oligopolistic* due to the presence of few large cloud service providers (e.g., Amazon or Microsoft) [16, 17]. In other words, a small number of larger service providers influence the prices. Oligopolistic prices

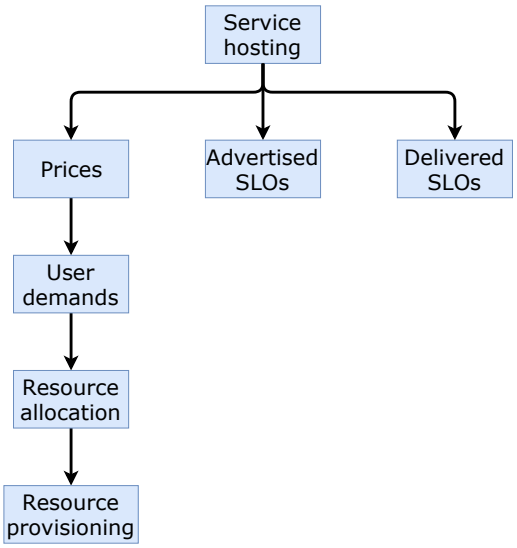



Figure 1.5: Relationship between pricing and resource provisioning.

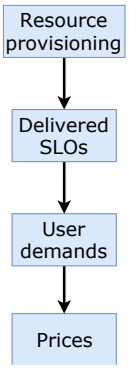



Figure 1.6: Relationship between resource provisioning and pricing.

are higher than competitive prices. Hence, fixed pricing neither maximizes the revenue of cloud service providers, nor it is fair towards the customers.

The cloud computing community is exploring an alternative pricing scheme called *dynamic pricing* [18–20] to address earlier mentioned fixed pricing issues. In this approach, resource prices reflect supply and demand of the resource. The higher the demand is, the higher the prices are. Most often dynamic pricing leads to efficient resource utilization and satisfaction of user demands [13]. There are some efforts from industry towards dynamic pricing. For instance, Amazon introduced the concept of *spot pricing*. In spot pricing, a user can specify maximum prices he is willing to pay, and instances are allocated until spot instance prices are within a maximum price. However, spot instance prices are not determined based on the market demand [20].

Generally, cloud service providers are profit driven. Hence, prices maximize the revenue for a service provider. This approach is beneficial in short-term but might drive away customers in the long term due to higher prices. Microeconomic theory addresses this issue by maximizing social welfare maximization [21]. In cloud computing, maximizing social welfare improves not only the overall system efficiency but also assures better user experience [13]. Furthermore, maximizing social welfare is apt for both public and private clouds [22]. Therefore, maximizing social welfare is beneficial for both the cloud service providers and the users.

There are three types of social welfare namely *utilitarian*, *egalitarian* or *max-min fairness* and *Nash Social Welfare* (NSW). In utilitarian, the aim is to maximize overall utility of customers and service providers. Most of the works in the Cloud regarding dynamic pricing (e.g., [7, 20, 23, 24]) are *utilitarian* and favor customers with higher utility. Conversely, the *egalitarian* goal is to maximize the minimum utility of the customers and rewards customers with lower utility. In literature [25], NSW is the *Pareto outcome* between utilitarian and egalitarian approaches. Moreover, NSW is *scale-free*, i.e., optimal allocation is independent of the scale of each customer's utility. Hence, maximizing NSW is appropriate in a cloud environment.

The market that maximizes the NSW is also called *Eisenberg-Gale* or *Fisher market* and is well studied in algorithmic game theory [26–29]. In this market setup, customers arrive with money, and the service providers allocate resources to maximize NSW. Most often, allocations are fractional. However, in the context of cloud computing, a fractional allocation is intangible. The usual approach is to round these fractional values to nearest integer. However, the resulting rounding difference (*integrality gap*) is unbounded for a market, i.e., the difference grows with the number of buyers [30].

Furthermore, *differential pricing* among resources should be possible for multiple reasons. First, the same VM can be more costly when provisioned during peak times (when

resources are more scarce) than off-peak hours. Second, differential pricing can be used as an incentive strategy to motivate users to spend not only environment-friendly offerings such as green energy based resources but also to distribute load across multiple data centers. Moreover, cloud resources often suffer from diminishing returns, i.e., often adding more copies of the same resource does not improve the utility for the buyer [31]. Furthermore, cloud customers are ready to pay differential prices for superior service [32].

State-of-the-art solutions in the cloud and edge computing currently do not offer capabilities for differential pricing [6,24] and ignores diminishing returns. Hence, discrete concave utilities such as the SPLC utility are proposed [33]. There is a solution based on stable polynomial approach [34], but it requires highly complex ellipsoidal algorithms. However, the cloud market requires fast market clearance [35]. Therefore, price computation should be quick. Auctions are another alternative for computing market equilibrium. Auction-based algorithms are combinatorial and run faster compared to ellipsoidal algorithms [36].

Auctioning is one of the dynamic pricing models widely used in cloud and edge computing environments due to its ability to discover the market value of resources without compromising economic efficiency [37]. Furthermore, auctioning helps to find a fair price in cases where both the seller (auctioneer) and the buyer do not know the actual value and estimates could be highly imperfect (price uncertainty) [38]. Auctions can be either *open-cry* or *sealed* [39] depending on the knowledge about other bidders. In open-cry auctions, the buyers observe other bidders and update their bids at every round. Sealed auctions are very popular in edge and cloud computing. In this auction, buyers submit sealed bids and sellers allocate the resources. The buyers might inflate or deflate their bids. Hence, auction mechanisms are proposed for encouraging truthful bidding by providing *incentives*. Hence, such auctions are *incentive compatible*. Most often, auction mechanisms assume prior information about buyers bidding values [40]. However, if a user draws a bid from a different probability distribution, then the obtained solution is not optimal [40, 41]. This is often the case in edge computing, where each user is different and may draw valuations from a different probability distribution.

Inherently, in cloud computing, customer demands are ad-hoc [13] and not independent and identically distributed [42]. The works mentioned above are *offline* algorithms – they have complete knowledge of input data. In contrast, for *online* algorithms data is revealed in parts. Recently, pricing based on online algorithms have captured interest [43–45], especially posted pricing [13, 46]. In posted pricing, customers appear sequentially, and the seller publishes prices. If prices are acceptable, then customers procure the resource; otherwise they reject it. Hence, this approach is alternatively known as *leave-it-or-take-it* pricing. Also, the online marketplace has enabled collection of a significant data for service providers. As a result, there are works based on online learning to compute prices [46]. However, these works maximize either utility or revenue maximizing.

In algorithmic game theory, the online versions of Fisher market where resources appear in each round are proposed [47, 48] but the assumption of resources appearing every round is contrary to the reality in the cloud and edge computing. These online algorithms are analyzed based on the adversarial models (worst-case inputs). There is a solution [49] to handle arbitrary customers and resources. However, the adversarial model is not only weak but also cannot guarantee integer allocation.

In summary, there are multiple needs regarding pricing resources. Firstly, an auction mechanism with no prior information about the buyer bids in edge computing. Secondly, an auction-based computationally efficient market that not only maximizes NSW with integer allocation of cloud resources but also enables differential pricing for SPLC utilities. And thirdly, an online market maximizes NSW for arbitrary customers and resources with stronger adversarial models for arbitrary resources and customers with guaranteed integer allocation.

### 1.2.2 Resource Provisioning

After the price computation, the subsequent step is to automate the provisioning of resources for the customer demanded services. Providers perform provisioning and scheduling optimizations due to substantial benefits in, for example, performance, and revenue or energy consumption [50]. One of the significant challenges in automated resource provisioning is to configure resources *optimally* to services as illustrated in Figure 1.3. Generally, a requested service is deployed as a combination of different components, where each component is assigned, a set of virtual instances and their resources.

Both industry and academia have addressed the problem of *when* to provision more resources (e.g., more component instances) for an application under high load. Many cloud service providers such as Google and Amazon employ threshold-based autoscaling strategies for provisioning. There are more sophisticated control-theory solutions [51–53] or systems based on empirical service modeling [54, 55]. Moreover, some approaches predict *how many* resources need to be provisioned based on time series analysis [56, 57]. Cloud providers exploit tools that address the question of *how to* provision these resources, such as Puppet, OpenStack Heat, Ansible, and Chef. Thus, the goal is to find the appropriate time to increase the resources to meet *service level objectives* (SLOs).

Often service providers overlook the question of what to provision and deploy all the components with a uniform size of VM (also called as *flavor*) during actual deployment, commonly known as *one-size-fits-all* and have faced several drawbacks. First, deciding VM size is non-trivial and requires the expertise of the service. Second, the configurations offered by VM are not uniform. Thus, translates to non-uniform load capacity. Finally, in case

of new paradigms such as NFV, services are composed dynamically from the available components. Most often, there are flows from one component to another. Consequently, there are dependencies among these components. In the one-size-fits-all approach, deployment ignores the dependencies. Consequently, service might not handle required load. In summary, one-size-fits-all results in either *over-provisioning* or *under-provisioning* and over-provisioning results in resource wastage. Conversely, under-provisioning degrades service performance [11]. Hence, the service provider may violate SLOs, consequently resulting in business and legal implications. This scenario holds true with multi-tier services as well. Hence, service providers need to address what to provision during service deployment based on arrival workload load.

As a first step to answer the question of *what to deploy*, it is essential to determine the performance cost and capacity of flavors since these parameters affect the SLOs directly. Generally, a VM specification provides information about the maximum capacity of the flavor. However, determining performance cost is non-trivial due to dynamically varying load [58]. Moreover, request arrivals are not restricted to specific probability distributions (e.g., Poisson). Queueing theory is widely applied for performance modeling of computer systems. The performance metrics (e.g., latency) are estimated from the customer/process arrivals and departures. Queueing models are widely prevalent in cloud computing as well [58, 59]. Most of the queueing models assume Markovian arrivals as it is computationally tractable, unlike generic queues. However, traffic in multi-tier systems is not Markovian [58]. There are some works such as [58] which model tiers using generic queue but could not consider processing capacity due to the computational intractability of generic queues.

It is well known that a complex service consists of multiple components and end-to-end performance costs need to be estimated from the individual components. Even if we estimate the end-to-end performance cost of a complex service, finding the *optimal* configuration for the complete service is non-trivial due to the *combinatorial* configuration space. For instance, consider a hypothetical complex service with three components and each supporting 4 configurations. The service can be deployed in $4 \times 4 \times 4 = 4^3$ ways and we have to find an *optimum* among this combinatorial search space.

Real complex services are composed of a large number of components and can be deployed on a variety of VMs. As a result, optimal configuration space of the service is vast. In addition, finding optimal configuration should be not only time efficient so that the solution can be deployed for edge applications with near-instant provisioning requirements [60] but also automated for integrating with cloud or edge orchestrators.

Therefore, there are multiple needs regarding automating resource provisioning. First, an answer to the "*what to deploy*" question that can be automated and used with tools such as Puppet, OpenStack Heat, Ansible or Chef. Moreover, the solution should be extended to applications which require near-instant provisioning [60] which is very common in edge

computing. Second, a model needs to be developed for determining the performance cost of VM for arbitrary arrivals and simultaneously accounting for processing capacities and computationally tractable.

For the remainder of this dissertation, we consider a complex service as a resource type during resource pricing.

## 1.3 Challenges

In this section, we present the challenges for pricing and resource provisioning in cloud computing.

### 1.3.1 Pricing

The ability to price resources based on supply-demand is the main strength of dynamic pricing. Consequently, prices are time-dependent and reflect the market demand of the resource. For instance, consider a hypothetical VM based on *x*64 hosted on Linux with 10 GB memory called *Tlarge*. In dynamic pricing, the *Tlarge* price is higher if the market demand for Tlarge is higher and vice versa. Section 1.1 emphasized the need for dynamic pricing in the cloud and presented multiple issues concerning pricing. However, we need to address the following challenges:

1. **Strategic behavior:** It is well-known that bidders are not allowed to modify bids after submission. Generally, customers inflate their demands and tighten deadlines anticipating improved SLOs [61]. Hence, customers might not bid truthfully. Moreover, sealed auctions are susceptible to *bidder collusion* – bidders collude among themselves and reduce payment to the auctioneer [62]. Hence, the auctioneer designs an incentive compatible auction mechanism to motivate truthful bidding. Therefore, the proposed pricing schemes should design an *incentive compatible* auction mechanism.

2. **No prior information:** Most of the auction mechanisms assume a prior distribution of user bids. However, if a user draws a bid from a different probability distribution, then the obtained solution/price is not optimal [40,41]. This is often the case in cloud and edge computing, where each user is different and may draw valuations from a different probability distribution. Hence, the auction based pricing mechanism should be adaptive to the probability distribution of user bids.

3. **Differential prices:** Social welfare maximization is one of the critical challenges for any service provider as it improves the overall system efficiency and provides a better user experience. In differential pricing, each customer pays a different price for the same resource types and consequently transfers user surplus (the difference between the money the customer is willing to pay and the actual money that is spent) to the service provider, eventually a higher revenue for the service provider. Therefore, differential pricing is inherently unfair towards some customers [63].

   As we know, NSW is Pareto outcome between utilitarian and egalitarian social welfares and the goal of this dissertation is to maximize NSW. Hence, the proposed pricing scheme should achieve NSW of all cloud customers in differential pricing.

4. **Integer allocation:** In today's cloud and edge datacenters, most of the resources are generally virtualized. There are plenty of works on market-based pricing. Most of these works compute fractional allocation. However, many cloud resources (e.g., VMs) cannot be allocated fractionally. Hence, the fractional values are either rounded up or down to the nearest integer. Unfortunately, the resulting rounding difference (*integrality gap*) is unbounded for a market, i.e., the difference between optimal objective value and rounded objective value grows with the number of buyers and resources [30, 34].

   Hence, the proposed pricing scheme should not round the fractional allocation to the nearest integer.

5. **Performance and scalability:** Maximizing NSW for indivisible items is *APX-hard* [64]. An optimization problem is *APX-hard* if it is in *NP* and allows polynomial-time approximation algorithms (PTAS) with approximation ratio bounded by a constant which implies the existence of an efficient algorithm to find market equilibrium within a fixed multiplicative factor of the optimal market equilibrium.

   Furthermore, cloud computing market is projected to become a low commodity market. Moreover, a large number of current service providers are offering edge resources. Therefore, the proposed pricing scheme should be time efficient and scale with customers and resources.

### 1.3.2 Resource Provisioning

In this subsection, we present the challenges for addressing *what to provision* problem in resource provisioning.

1. **Estimating performance cost:** Generally, service providers guarantee SLOs for cus-

tomers. Hence, the optimal configuration of the complex service should not violate promised SLOs. Otherwise, it leads to business and legal implications. A complex service is composed of heterogeneous components. Each component offers different functionality and has a different capacity. Hence, it is vital to estimate the performance cost of component individually. However, estimating the performance cost is non-trivial for real cloud systems because of the varying workload and customer arrivals. In other words, the customer arrivals are not restricted to a specific distribution.

There are queueing theory-based approaches to estimate performance cost. Most of these approaches assume exponential arrivals. Although there are some approaches based on generic distribution, they fail to take processing capacity of servers into account due to the intractability of generic distribution based queueing models. In summary, any performance model of a complex service should satisfy following properties for finding optimal configurations:

- Functionality independence.
- Adaptive to incoming workload.
- Support generic distributed arrivals and departures.
- Account for the processing capacity of each component.
- Multi-tier aware.

The above properties ensure that proposed resource provisioning algorithm does not violate agreed SLOs.

2. **Performance and scalability:** More and more customers are moving towards cloud and edge. Furthermore, recently service providers are introducing various types of complex services, such as analytics and virtual networking. As a result, complex services are composed of a large number of heterogeneous components. Cloud or edge orchestrators perform the deployment of complex services. Moreover, few edge applications require near-instant provisioning. As a result, orchestrator's have to find the optimal configuration within a small amount of time irrespective of the service.

   Hence, the proposed resource provisioning algorithm has multiple requirements. First, the solution should scale with the number of components. Second, the time required to find an optimal configuration should be limited for applications that require instant provisioning. Finally, the solution should be able to integrate with tools, such as Chef and Heat so that provisioning solution can be integrated with the orchestrator.

### 1.3.3 Summary

In summary, requirements for an automated resource pricing and provisioning solution to address above challenges are presented below:

(i) Designing an *incentive* compatible auction mechanism for edge and cloud resources to motivate truthful bidding in sealed auctions. Furthermore, the proposed mechanism should be adaptive to an arbitrary probability distribution of bids.

(ii) Finding differential prices for the resources keeping NSW maximized. The proposed solution should scale with a large number of different resource types and be time efficient. Moreover, the resource allocation should be an integer.

iii) The performance estimation model of a complex service should be adaptive to incoming workload, be multi-tier aware and be component functionality independent. Furthermore, it should estimate performance cost based processing capacity of the component and arbitrary customer arrivals and departure.

(iv) The proposed solution should have the ability to work with tools seamlessly (e.g., Chef and Heat) so the decision of finding optimal configuration can be automated inside an edge or cloud orchestrator.

## 1.4 Contributions

In this dissertation, we propose `ARPP` to perform automated resource pricing and provisioning for cloud and edge resources as shown in Figure 1.7.

The cloud and edge users arrive at the cloud service provider. The service provider provides three pricing options to the user. First, a user can submit a sealed bid for edge resources. Second, users can procure resources from an auction market which maximize NSW and supports differential pricing without violating user's budget. Finally, a user shall be able to procure resources from an online Fisher market by paying prices at current time instance. Once the prices are computed, the service provider provisions the resources by finding deployment configurations that maximize the system utilization. This approach not only maximizes NSW of users but also maximizes the overall resource utilization which eventually improves market competitiveness and resource utilization of the service providers.

The salient features of `ARPP` pricing approaches are:

(a) ***Robust Auction for Edge Resource Allocation*** **(RAERA)**: `RAERA` is a robust optimization-based auction mechanism for multi-item auctions for use with edge computing resources. The users submit sealed bids for the resources. `RAERA` leverages

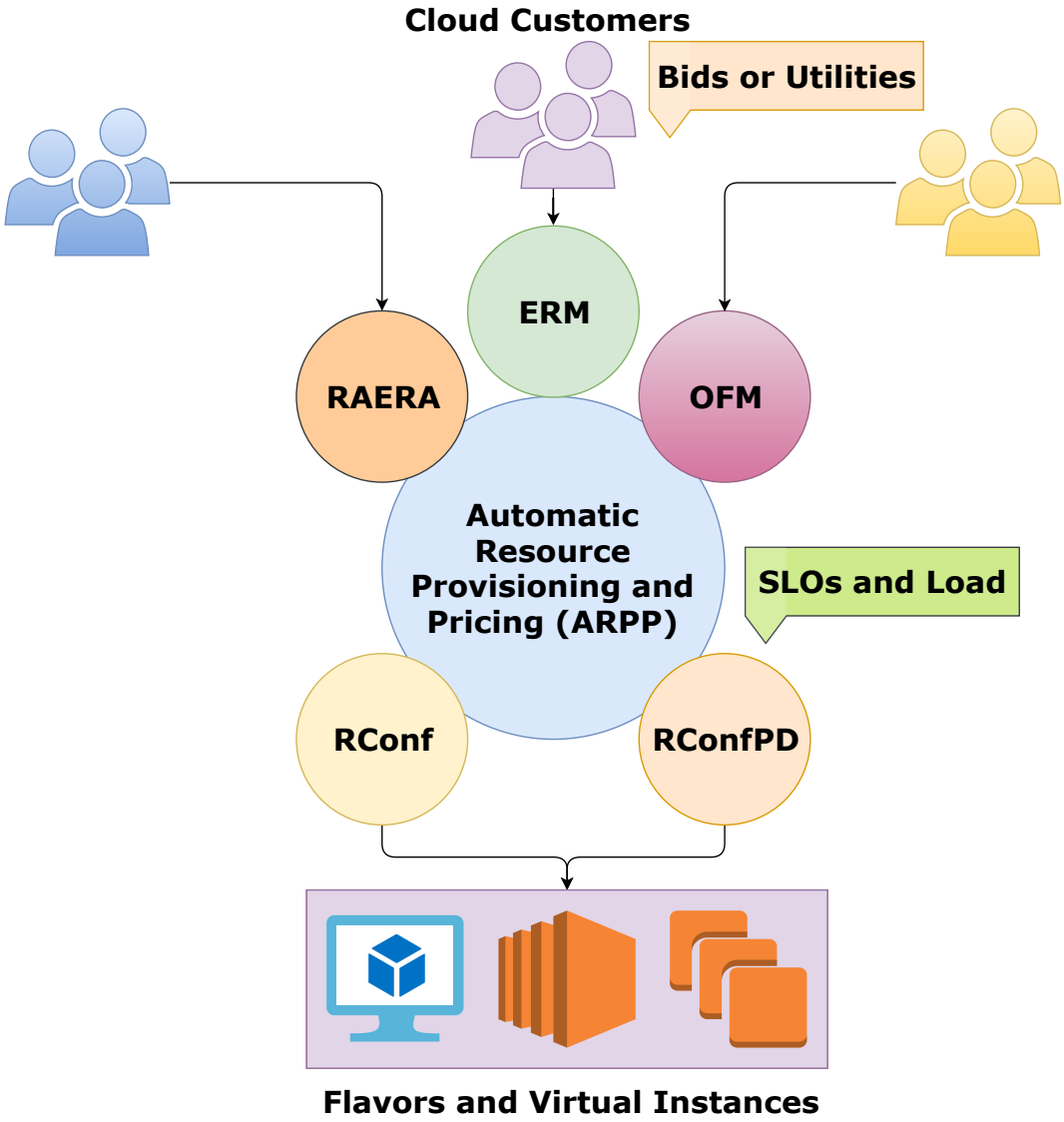



Figure 1.7: ARPP for pricing and provisioning

historical bid data and determines the winner. This strategy guarantees the profit for service providers during price uncertainty, and also calculates the reserve prices based on historical data and leverages this is before the winner determination. As a result, it guarantees the profit for service providers during price uncertainty.

RAERA satisfies *incentive compatibility* (truthful bids have higher utility for the bidder compared to non-truthful bids) and *individual rationality* (bidders do not derive negative utility for truthful bids). Therefore, RAERA can set a time-dependent fair price that benefits both buyers and sellers.

(b) ***Edge Resource Market*** **(ERM)**: ERM is an auction-based Fisher market that offers multiple resource types and buyers with SPLC utility. ERM guarantees $(1+\varepsilon)$ approximation. Further, ERM ensures *differential prices* and integer resource allocation without violating the $(1+\varepsilon)$ market equilibrium.

ERM is the first auction based approach to exploit SPLC utilities for capturing diminishing returns, which allow the price differential to vary depending on utility, budget and resource types. Experimental evaluation shows ERM offers orders of magnitude more revenue for the market provider than state-of-the-art approaches. Furthermore, ERM scales well with increasing number of buyers and resource types.

(c) ***Online Fisher Market*** **(OFM)**: OFM is an online Fisher market for cloud and edge resources. OFM is the first approach that computes the prices for current time instant based on previous instant data. The actual inputs such as buyer utility and resources offered are revealed only after current prices are computed.

OFM maximizes NSW and performs integer allocation. Moreover, it achieves a balance between the well-behavedness and worst-case inputs. The experimental results on both real and emulated datasets demonstrate the convergence and time efficiency of OFM.

Once the prices are computed, it is necessary to provision resources for maximizing the system utilization without affecting SLOs. In this dissertation, we propose RConf and RConfPD as a part of ARPP provisioning to address the question of *what to provision*. The prominent features of ARPP provisioning are:

(a) ***Robust Configuration*** **(RConf)**: We formalize the problem of finding the optimal configuration for complex service as *Virtual Configuration Problem* (VCP) and reduce it to *Multiple choice Multidimensional Knapsack Problem* (MMKP). In this dissertation, we propose RConf as a *near-optimal* solution for **VCP**.

RConf takes advantage of robust queueing theory-based model to predict performance cost of the configurations. We believe that the proposed model is the first approach for

predicting performance cost of a component with a generic arrival and departure process and simultaneously considering processing capacity unlike state of the art solutions. The experiments on Amazon EC2 demonstrate the prediction efficiency of the model. `RConf` finds an optimal solution, i.e., finds the best available flavors for each service component under SLO.

(b) ***Robust Configuration Primal-Dual*** **(RConfPD)**: `RConf` requires optimization solvers to find an optimal solution. However, these solvers have exponential time complexity. Hence, they are not appropriate for applications with near-instant provisioning requirement.

`RConfPD` is a primal-dual based algorithm with $1+\varepsilon$ approximation guarantee for **VCP** and trades off optimality against computational complexity. We believe that `RConfPD` is the first primal-dual approximation solution for MMKP. The experiments on Amazon EC2 and simulation demonstrate the superiority of `RConf` and `RConfPD` regarding overall resource utilization and resource usage over conventional approaches.

## 1.5 Dissertation Overview

The remainder of this dissertation is organized as follows: in Chapter 2, the basic concepts of linear programming, duality theory, microeconomic theory and robust queueing theory are explained. A comprehensive review of related work on both resource pricing and provisioning is presented in Chapter 3. The review reveals the need for an `ARPP` solution. The `ARPP` sealed bid auction called RAERA is presented in Chapter 4. The `ARPP` module for differential pricing is proposed in Chapter 5. The online `ARPP` based on Fisher market is presented in Chapter 6. Chapter 7 presents `ARPP` resource provisioning and Chapter 8 presents the evaluation results of `ARPP` resource provisioning.

# Chapter 2

# Background

This chapter starts with providing the theoretical concepts for comprehending ARPP solutions proposed in this dissertation. First, we introduce linear programming and duality theory necessary for primal-dual based approximation algorithms. Furthermore, we briefly introduce *Karush-Kuhn-Tucker* conditions for a generic function. Secondly, we present fundamental concepts in microeconomic theory required for Fisher market. Finally, we briefly provide an overview of *robust queueing theory* essential for understanding the performance model proposed as a part of the answer to the question, *what to provision*.

## Contents

## 2.1 Optimization theory

In this section, we introduce basic concepts in linear programming, specifically duality based on [65] comprehending Chapter 7. The readers can explore [65] for an in-depth explanation of linear programming.

### 2.1.1 Linear Programming and Duality

Given a cost vector $\boldsymbol{c} = (c_1, \ldots, c_n)$ of $n$-dimensional vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ and our goal is to minimize the product $\boldsymbol{c}^\intercal \cdot \boldsymbol{x} = \sum_{i=1}^{n} c_i x_i$, also called as *linear cost function* subject to a set of linear equality and inequality constraints. Then, the corresponding linear program is given

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \boldsymbol{c}^\intercal \cdot \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{a_i}^\intercal \cdot \boldsymbol{x} \geq b_i, \quad i \in F_1, \\ & \boldsymbol{a_i}^\intercal \cdot \boldsymbol{x} \leq b_i, \quad i \in F_2, \\ & \boldsymbol{a_i}^\intercal \cdot \boldsymbol{x} = b_i, \quad i \in F_3, \\ & x_j \geq 0, \quad j \in B_1, \\ & x_j \leq 0, \quad j \in B_2. \end{aligned} \tag{2.1.1}$$

In the above Eq. (2.1.1), $F_1, F_2$, and $F_3$ are some finite sets used to construct a constraint based on the n-dimensional vector $\boldsymbol{a_i}$ and a scalar $b_i$. Similarly, the sets $B_1$ and $B_2$ constrains the variables $x_j$ as either non-negative or non-positive. The variables $x_1, \ldots x_n$ are popularly known as *decision variables*. The vector $\boldsymbol{x}$ satisfying all the constraints of Eq. (2.1.1) is called as a *feasible solution* or *feasible vector*.

There can be multiple feasible vectors for a linear program. Let $\boldsymbol{x}^\star$ be the feasible vector that minimizes the objective function, i.e., $\boldsymbol{c}^\intercal \cdot \boldsymbol{x}^\star \leq \boldsymbol{c}^\intercal \cdot \boldsymbol{x}$ for all feasible $\boldsymbol{x}$. Then, $\boldsymbol{x}^\star$ is called as an *optimal feasible solution* or *optimal solution*.

Suppose, there are $m$ constraints indexed by $i$. Let $\boldsymbol{b} = (b_1, \ldots, b_n)$ and let $\boldsymbol{A}$ be a $n \times m$ matrix. The row vectors $\boldsymbol{a_1}^\intercal \ldots \boldsymbol{a_m}^\intercal$ form the row of matrix $\boldsymbol{A}$.

$$\boldsymbol{A} = \begin{bmatrix} \ldots & \boldsymbol{a_1}^\intercal & \ldots \\ & \vdots & \\ \ldots & \boldsymbol{a_m}^\intercal & \ldots \end{bmatrix}$$

We can represent above constraints as $\boldsymbol{Ax} = \boldsymbol{b}$. Then, Eq. (2.1.1) can be rewritten as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \boldsymbol{c}^{\mathsf{T}} \cdot \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{Ax} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq 0. \end{aligned} \tag{2.1.2}$$

The linear program of Eq. (2.1.1) is said to be in the *standard form* (Eq. (2.1.2)). The procedure to convert the generic form to standard form can be found in [65].

We can solve linear programs in multiple ways. However, duality theory is widely used. Duality theory is based on the Lagrange multiplier method used in calculus [66] to minimize a function with equality constraints.

Consider an example function as given [65]

$$\begin{aligned} \underset{x,y}{\text{minimize}} \quad & x^2 + y^2 \\ \text{subject to} \quad & x + y = 1. \end{aligned} \tag{2.1.3}$$

In Lagrange multiplier method, Eq. (2.1.3) is first transformed to an unconstrained minimization problem by introducing a Lagrange multiplier for each constraint. Let $p$ be the Lagrange multiplier for constraint $x+y=1$. The Lagrange $\mathscr{L}(x,y,p)$ of Eq. (2.1.3) is given by

$$\mathscr{L}(x,y,p) = x^2 + y^2 + p(1-x-y) \tag{2.1.4}$$

The Lagrangean $\mathscr{L}$ is also called as *Lagrange dual* is minimized over $x$ and $y$ while keeping $p$ fixed i.e., $\dfrac{\partial \mathscr{L}}{\partial x} = 0$ and $\dfrac{\partial \mathscr{L}}{\partial y} = 0$. The Lagrangean $\mathscr{L}$ achieve optimality when $x = y = \dfrac{p}{2}$. The constraint of Eq. (2.1.3) implies that $p = 1$. Hence, the optimal solution of Eq. (2.1.3) is $x = y = \dfrac{1}{2}$.

The Lagrange multiplier method can be summarized as follows: we associate a Lagrange multiplier for each constraint which is called popularly as *price* and allow constraint violation by converting to an unconstrained optimization problem. We find prices such that the optimal solution of both constrained and unconstrained optimization problems are equal. In other words, we find the prices such that constraints do not affect the optimal cost.

Linear programming uses the Lagrange multiplier method. Hence, associates price for each constraint. The goal is to find prices such that constraints do not affect the optimal cost

that eventually leads to a new linear program called *dual* and the original linear program called *primal*.

The dual of the standard linear program (Eq. (2.1.2)) is given by

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \boldsymbol{p}^{\mathsf{T}} \cdot \boldsymbol{b} \\ \text{subject to} \quad & \boldsymbol{p}^{\mathsf{T}} \boldsymbol{A} \leq \boldsymbol{c}^{\mathsf{T}}. \end{aligned} \tag{2.1.5}$$

The dual of a minimization problem is a maximization problem and vice versa. The procedure of finding dual can be found in [65].

Weak and strong duality theorems describe the relationship between objective functions of primal and dual. The proofs can be found in [65].

**Theorem 2.1 (Weak duality)** If $\boldsymbol{x}$ and $\boldsymbol{p}$ are feasible solutions of *primal* and *dual* respectively in standard form, then $\boldsymbol{p}^{\mathsf{T}}\boldsymbol{b} \leq \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x}$

**Theorem 2.2 (Strong duality)** If a primal has an optimal solution, and the corresponding dual also has an optimal solution then the respective optimal costs are equal.

Theorem 2.1 and Theorem 2.2 is known as *weak* and *strong* duality theorems respectively. The weak duality theorem implies that cost of primal and dual are bound to each other. Conversely, strong duality theorem implies that the cost of primal and dual is equal at optimality. Furthermore, this leads to an important relationship between primal and dual optimal solution called *complementary slackness* as defined in the following theorem:

**Theorem 2.3 (Complementary slackness)** If $\boldsymbol{x}$ and $\boldsymbol{p}$ are feasible solutions of *primal* and *dual* respectively in standard form. If $\boldsymbol{x}$ and $\boldsymbol{p}$ are *optimal* solutions only iff:

$$\begin{aligned} p_i(a_i^{\mathsf{T}} - b_i) &= 0, \forall i \\ (c_j - \boldsymbol{p}^{\mathsf{T}}\boldsymbol{A}_j)x_j &= 0, \forall j \end{aligned}$$

Theorem 2.3 is the cornerstone for designing primal-dual approximation algorithms [67].

### 2.1.2 *Karush-Kuhn-Tucker* Conditions

In the previous section, Lagrange multiplier method is applied to a linear program with equality constraints. *Karush-Kuhn-Tucker* (KKT) conditions extend Lagrange multiplier

method for a generic function with inequality constraints. We briefly introduce KKT conditions required for comprehending Section 2.2. The readers can refer [68] for an in-depth discussion on both convex programming duality and KKT conditions.

Consider a generic maximization function

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad \forall i = 1, \cdots, m, \\ & g_j(x) \leq 0, \quad \forall i = 1, \cdots, n \end{aligned} \tag{2.1.6}$$

Let $\alpha_i$ be the Lagrangean multiplier associated with the equality constraint $h_i(x)$ in Eq. (2.1.6). Similarly, $\mu_j$ be the Lagrangean multiplier associated with inequality constraint $g_j(x)$. The *Lagrangean dual* $\mathscr{L}$ is given by

$$\mathscr{L}(x, \alpha, \mu) = f(x) - \sum_{i=1}^{m} \alpha_i h_i(x) - \sum_{j=1}^{n} \mu_j g_j(x) \tag{2.1.7}$$

Let $\nabla_x$ be the gradient operator of the function i.e., $\nabla_x f(x)$ represents the gradient of function $f(x)$ with respect to $x$. Let $x^\star$ be the optimal solution of Eq. (2.1.6), then KKT conditions are:

(i) Stationary: $\nabla_x f(x^\star) = \sum_{i=1}^{m} \alpha_i \nabla_x h_i(x^\star) + \sum_{j=1}^{n} \mu_j \nabla_x g_j(x^\star)$ i.e., $\dfrac{\partial \mathscr{L}}{\partial x} = 0.$

(ii) Equality constraints: $\nabla_\alpha f(x^\star) + \sum_{i=1}^{m} \nabla_\alpha \alpha_i h_i(x^\star) + \sum_{j=1}^{n} \mu_j \nabla_\alpha g_j(x^\star) = 0$ i.e., $\dfrac{\partial \mathscr{L}}{\partial \alpha} = 0.$

(iii) Complementary slackness: $\mu_j g_j(x) = 0, \forall j = 1, \cdots, n.$

## 2.2 Microeconomic theory

The unique characteristic of microeconomic theory is the ability to model economic activity as an interaction of economic agents with private information [21]. Individual decision making is not only a significant concept but also basis for most of the analysis in microeconomic theory. In individual decision making, the goal is to choose outcomes. The decision-making model may be either *preference based* or *choice based* depending on the relationship among the outcomes. In preference based models, preference relation exists among the outcomes based on the rationality axioms; i.e., the decision maker prefers some outcome over the other. In choice-based models, the decision maker makes choice a based on the consistency

criterion. Our work is based primarily on preference-based individual decision making. Hence, in this section, we present fundamental concepts of preference-based individual decision making presented in [21].

In preference-based models, preference relation exists among the outcomes. Let $\mathscr{X}$ denote the outcome set and $\succsim$ denote the preference relation between outcomes. Let $a$ and $b$ be two outcomes in $\mathscr{X}$ i.e., $a,b \in \mathscr{X}$. If a decision maker prefer $a$ over $b$, then it is denoted as $a \succsim b$.

**Definition 2.1 (Rational)** If preference relation $\succsim$ is said to be *rational* if the following properties are satisfied:

- *Completeness*: $\forall a,b \in \mathscr{X}$, we have $a \succsim b$, or $b \succsim a$ or both. In other words, preference relation exists with all the outcomes in the set.
- *Transitivity*: $\forall a,b,c \in \mathscr{X}$, If $a \succsim b$ and $b \succsim c$, then $a \succsim c$.

In economics, preference relations are described using a function which assigns a numerical value to each outcome. This function is known as *utility function* or *utility*. Formally, a utility function is defined as follows:

**Definition 2.2 (Utility function)** A function $u : \mathscr{X} \to \mathbb{R}$ is a utility function representing preference relation $\succsim$ if

$$\forall a,b \in \mathscr{X}, a \succsim b \iff u(a) \geq u(b)$$

In microeconomic theory, the *market economy* is one of the approaches to address the fundamental issue of production organization and commodity allocation [21]. *Market equilibrium* is an outcome in a market economy such that agents maximize their utility without violating budget constraints and market clears (complete allocation of goods).

Consider a market $\mathscr{M}$ with $n$ buyers and $m$ *divisible* goods and represented by the sets $B$ and $G$ respectively, i.e., $|B| = n, |G| = m$. Let the sets $B$ and $G$ be indexed by $i$ and $j$, i.e., $i$ represents $i^{th}$ buyer and $j$ represents $j^{th}$ goods. Let $b_i$ be the maximum budget of the $i^{th}$ buyer. Let $u_{ij}$ be the utility derived by the $i^{th}$ buyer for $j^{th}$ good. Also, $x_{ij}$ be the fraction of $j^{th}$ goods allocated to the $i^{th}$ buyer. When the utilities are linear, the total utility derived by $i^{th}$ buyer is $\sum_{j=1}^{m} u_{ij}x_{ij}$ and represented by $U_i$.

The objective function of market $\mathscr{M}$ should satisfy following conditions [26]:

- The optimal allocation should be invariant of utility scaling. In other words, if a utility of a buyer is scaled, then optimal allocation should be unaltered.
- If the budget of a buyer $i$ is split among two new buyers with same utility, then the sum of optimal allocation of new buyers should be the same as optimal allocation of buyer $i$.

The geometric mean of the utilities satisfies the above properties [26]. As we know, NSW is the geometric mean of the utilities of all the agents. The objective of market $\mathscr{M}$ is to maximize NSW. Hence, we have following objective function:

$$\text{maximize} \quad \prod_{i \in B} \left( U_i^{b_i} \right)^{\frac{1}{\sum_i b_i}} \tag{2.2.1}$$

In Eq. (2.2.1), if we normalize total budgets to 1 (i.e., $\sum_i b_i = 1$) and take a logarithm of the objective, then Eq. (2.2.1) is transformed to the following:

$$\text{maximize} \quad \sum_{i \in B} b_i \log U_i \tag{2.2.2}$$

The convex program for maximizing NSW is given by

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{i \in B} b_i \log U_i \\ \text{subject to} \quad & U_i = \sum_{j=1}^{m} u_{ij} x_{ij}, \quad \forall i \in B, \\ & \sum_{i=1}^{n} x_{ij} \leq 1, \qquad \forall j \in G, \\ & x_{ij} \geq 0, \qquad \forall i \in B, j \in G. \end{aligned} \tag{2.2.3}$$

The convex program Eq. (2.2.3) is popularly called as the *Eisenberg-Gale* convex program [26]. The market $\mathscr{M}$ satisfying Eq. (2.2.3) is called the *Fisher* or *Eisenberg-Gale* market. Let $p_j$ be the price of the good $j$. Applying KKT conditions, the optimal price of a $j^{th}$ good is given by $p_j = \dfrac{b_i u_{ij}}{U_i}$. The derivation can be found in Appendix A.1.

Before presenting the theorem, we define a rational equilibrium [33, 69].

**Definition 2.3 (Rational equilibrium)** If the utility, budget, allocation, and prices of a market $\mathscr{M}$ are rational then the market $\mathscr{M}$ is said to have a *rational equilibrium*.

Rational equilibrium implies that the utilities, budget, allocation, and prices can be written in polynomially many bits [33].

We present the equilibrium of Fisher market theorem proposed in [69].

**Theorem 2.4 (Fisher market pricing)** The Fisher market of buyers with linear utility, if there is a buyer $i$ with $u_{ij}$ for every goods $j$ (potential buyer for every good) then,

(i) Rational equilibrium exists.
(ii) The set of equilibrium allocations is convex.
(iii) Both equilibrium allocation and prices are unique.

**Proof** By KKT condition, we have $p_j \geq \frac{b_i u_{ij}}{U_i} \geq 0$ (Eq. (A.1.3)). Also, there is a buyer for every goods $j$. Hence, the prices $p_j > 0, \forall j \in G$. The complementary slackness (Eq. (A.1.2)) implies that $\sum_{i=1}^{n} x_{ij} = 1, \forall j \in G$. In other words, all the goods are completely allocated.

For $x_{ij} > 0$, $p_j = \frac{b_i u_{ij}}{U_i}$. Therefore, $p_j x_{ij} = x_{ij} \frac{b_i u_{ij}}{U_i}$. Summing over successful allocation of buyer $i$, we get $\frac{b_i \sum_{j=1}^{m} u_{ij}}{U_i} = \sum_{j=1}^{m} p_j x_{ij}$. Since utility is linear, $\sum_{j=1}^{m} u_{ij} = U_i$. Hence,

$$b_i = \sum_{j=1}^{m} p_j x_{ij}$$

In other words, the buyer $i$ spends his budget completely.

The utility and the budgets are rational. Also, in the market there is a potential buyer, $U_i \neq 0, \forall i \in B$. Hence, the prices are rational.

At equilibrium, allocation $x_{ij}$ is the solution of the convex program (Eq.2.2.3). Hence, equilibrium allocations form a convex set.

The logarithmic function is strictly concave. Therefore, the utility derived by each buyer should be the same across equilibria. Moreover, the buyer has to spend his budgets, and that implies the existence of only one equilibrium.

## 2.3 Robust Queueing Theory

In this section, we briefly introduce robust queueing theory proposed in [70]. As a prerequisite, first, we briefly introduce robust optimization and queueing theory.

### 2.3.1 Robust Optimization

The goal of optimization is to *maximize* or *minimize* given objective function and data. Correspondingly the solution is called *optimal*. In reality, most often data are unreliable due to errors such as measurement and estimation. Hence, data is uncertain due to errors. However, a small perturbation in data affects optimality [71, 72]. Stochastic optimization methods incorporate probability distribution of uncertainty during optimization to address uncertainty. However, obtaining uncertainty probability distribution in the real world is very hard and in few cases not possible [73]. Conversely, *Robust Optimization* (RO) approach bound the data within a deterministic set called *uncertainty set* during optimization [71]. Moreover, RO does not require any knowledge of probability distribution, unlike stochastic optimization.

We present the generic robust optimization formulation of [72]. Consider an objective function $f(x)$ and goal is to optimize under $m$ constraints $h_i(x,u_i) \leq 0$ with uncertainty parameters $u_i$, then,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & h_i(x,u_i) \leq 0, \quad \forall i = 1,\cdots,m, u_i \in \mathscr{U}. \end{aligned} \tag{2.3.1}$$

In the above formulation, $x \in \mathbb{R}^n$ is the decision vector and $f, h_i : \mathbb{R}^n \to \mathbb{R}$. Moreover, uncertainty parameters$u_i$ take arbitrary values from uncertainty set $\mathscr{U}_i$, i.e., $u_i \in \mathscr{U}_i \subseteq \mathbb{R}$. The aim of Eq. (2.3.1) is to find *optimal* decision vector $x^\star$ that not only minimize the objective function but also unaffected by all the disturbances $u_i$ within $\mathscr{U}_i$. Let us assume that there is an uncertainty set $\mathscr{U}$ such that $\mathscr{U} = \mathscr{U}_1 \times \cdots \times \mathscr{U}_m$. Hence, $(u_1,\cdots,u_m) \in \mathscr{U}$. Intuitively, $\mathscr{U}_i$ is a projection of $\mathscr{U}$ along corresponding dimensions [71].

Thus, RO guarantees solution feasibility and optimality against all instances of the parameters within the uncertainty set at the expense of computational overhead. The interested readers can refer [71, 72] for in-depth coverage of RO.

### 2.3.2 Queueing Theory

Queuing theory is widely used for modeling performance of the computer systems [74]. Figure 2.1 illustrates the general queueing system.

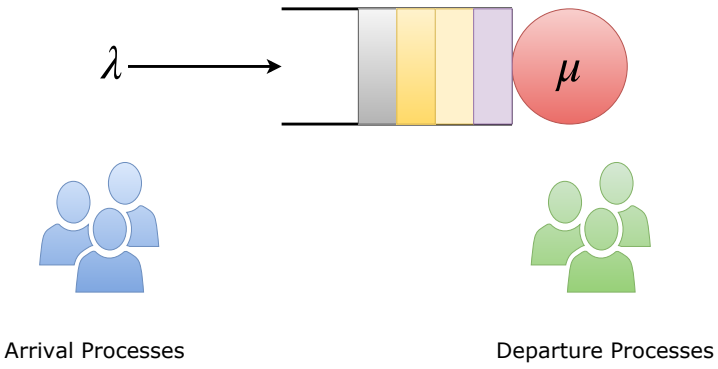



Figure 2.1: Illustration of queueing systems.

In a queue, the customers arrive at the service center. If the server is free, they are served immediately. Otherwise, they wait in the queue. Similarly, we can observe similar behavior in various resources, such as CPU and disk. For instance, jobs arrive at CPU scheduler and executed if CPU is available. Otherwise, jobs wait in the queue.

Queueing models provide two crucial insights. First, the performance of the system and second, capacity provisioning of the systems. These two insights are of immense value for a system designer. We introduce basic terminologies of queueing models.

First, we define a stochastic process as given in [75].

**Definition 2.4 (Stochastic Process)** A stochastic process is a group of random variables $\{X(t)|t \in T\}$, defined on a given probability space, indexed by the parameter $t$ which varies over the set $T$.

The values of $X(t)$ are called **states**, and the set of all possible values of $X(t)$ is called a **state space** of the process. Also, $T$ called an **index set**.

Consider a queue with $m$ parallel servers. Let $A_n$ and $X_n$ be the arrival and service time of $n^{th}$ job respectively. Let $T_n$ be the interarrival time and computed as $T_n = A_n - A_{n-1}$. Let $F_A$ be the probability distribution from which the interarrival times are drawn. Similarly, $F_X$ be the probability distribution of the service time. Using standard Kendall's notation [76], we can denote the queue as $F_A/F_S/m$. Table 2.1 shows the standard symbols used in queueing

theory literature. Consider a hypothetical notation $G/M/3$. This notation denotes a queue with three servers where the interarrival times are generally distributed while the service times are exponentially distributed.

| Symbol | Distribution type |
|---|---|
| M | Exponential distribution |
| D | Deterministic or constant arrival or service |
| G | General distribution |

Table 2.1: Common arrival and service distribution and their symbols used in queueing theory literature.

Let $\lambda$ be the rate of arrival process, i.e., number of arrivals per unit time. Similarly, $\frac{1}{\mu}$ be the mean service time at a server. The *utilization* of the queue denoted by $\rho$ is calculated as

$$\rho = \frac{\lambda}{m\mu} \tag{2.3.2}$$

The queue is said to be *stable* if $0 \leq \rho < 1$. Let $w_n$ be the waiting time of the $n^{th}$ job (i.e., time a job waits to be served). Moreover, let $\tau_n$ be *system time*, i.e., the end-to-end time spent by the job from arrival to departure and defined as

$$\tau_n = w_n + X_n \tag{2.3.3}$$

In other words, system time is the sum of *waiting time* and *service time*. Let $\overline{\tau}$ be the average system time. Then, according to *Little's result* [76], the number of jobs in the queue $N_q$ is given by

$$N_q = \lambda \overline{\tau} \tag{2.3.4}$$

Table 2.2 summarizes the queueing notations described in this section.

### 2.3.3 Robust Queueing Theory

If the arrival and departure process are exponentially distributed (Markovian) or deterministic, then we can derive the closed form expressions for system time and waiting time [76]. Hence, these queueing models are widely used due to this tractable behavior. However, if the arrivals and departures are arbitrarily distributed, then the generic method of analyzing steady state [77] is intractable [70]. Hence, most of the analysis is limited to Markovian or deterministic.

| Symbol | Description |
|---|---|
| $A_n$ | Arrival time of $n^{th}$ job |
| $X_n$ | Service time of $n^{th}$ job |
| $T_n$ | Interarrival time of $n^{th}$ job |
| $w_n$ | Waiting time of $n^{th}$ job |
| $\tau_n$ | System time of $n^{th}$ job |
| $\lambda$ | Arrival rate |
| $\mu$ | Service rate |
| $m$ | Number of servers |
| $\rho$ | Queue utilization |
| $N_q$ | Number of jobs in the queue |

Table 2.2: Summary of queueing theory notations described in this section.

Bandi *et al.* [70] apply robust optimization [71] to derive performance bounds for G/G/m queue. We can classify arrival and departure processes as heavy-tailed and non heavy-tailed depending on the bound of processes tail. In heavy-tailed distributions, a tail is not exponentially bounded. Authors construct uncertainty using the central limit theorem. According to the central limit theorem, the sum of independent and identically distributed random variables approach a normal distribution. For heavy-tailed processes, the sum converges to a stable distribution [78]. Let $\alpha_a$ and $\alpha_s$ be the tail coefficient of arrival and departure process respectively. Let interarrival times $T_1, T_2, \cdots, T_n$ be independent and identically distributed with mean $\frac{1}{\lambda}$. Similarly, service times $X_1, X_2, \cdots X_n$ be independent and identically distributed with mean $\frac{1}{\mu}$. Also, $\sigma_a$ and $\sigma_s$ be the standard deviation of interarrival and service times respectively. According to central limit theorem, as $n \to \infty$, then $\frac{\sum_{i=1}^{n} T_i - \frac{n}{\lambda}}{\sigma_a \sqrt[2]{n}}$ and $\frac{\sum_{i=1}^{n} X_i - \frac{n}{\mu}}{\sigma_s \sqrt[2]{n}}$ are asymptotically standard normal. Hence,

$$\sum_{i=1}^{n} T_i - \frac{n}{\lambda} \geq -\Gamma_a \sqrt[2]{n} \tag{2.3.5}$$

$$\sum_{i=1}^{n} X_i - \frac{n}{\mu} \leq \Gamma_s \sqrt[2]{n} \tag{2.3.6}$$

In Eq. (2.3.5) and (2.3.6), $\Gamma_a$ and $\Gamma_s$ are the variability parameter of arrival and departure

process. As we know that a standard normal $Z$ satisfies $\mathbb{P}(Z \leq 2) \approx 0.975$ and $\mathbb{P}(Z \leq 3) \approx 0.995$. Hence, $\Gamma_a$ and $\Gamma_s$ are chosen with higher $\mathbb{P}(Z)$. Authors extend the results to heavy distribution as well [70].

Hence, the uncertainty sets for arrival $\mathscr{U}_a$ and departure process $\mathscr{U}_s$ is given by

$$\mathscr{U}_a = \left\{ (T_1, \cdots, T_n) \middle| \frac{\sum_{i=1}^{n} T_i - \frac{n}{\lambda}}{n^{\frac{1}{\alpha_a}}} \geq -\Gamma_a \right\} \tag{2.3.7}$$

$$\mathscr{U}_s = \left\{ (X_1, \cdots, X_n) \middle| \frac{\sum_{i=1}^{n} X_i - \frac{n}{\mu}}{n^{\frac{1}{\alpha_s}}} \leq \Gamma_s \right\} \tag{2.3.8}$$

**Theorem 2.5 (Worst case bound for system time)** The worst case bound for system time $S_n$ of an $n^{th}$ job of m-server queue with $\boldsymbol{T} \in \mathscr{U}_a, \boldsymbol{X} \in \mathscr{U}_s, \alpha_a \neq \alpha_s$ such that $\rho < 1$ and $\overline{\alpha} = \min(\alpha_a, \alpha_s)$,

$$S_n \leq \frac{\overline{\alpha} - 1}{\overline{\alpha}^{\frac{\overline{\alpha}}{(\overline{\alpha}-1)}}} \frac{\lambda^{\frac{1}{(\overline{\alpha}-1)}} (\Gamma_a + \Gamma_s / m^{1/\overline{\alpha}})^{\overline{\alpha}/(\overline{\alpha}-1)}}{(1-\rho)^{\frac{1}{\overline{\alpha}-1}}} + \frac{m}{\lambda}$$

**Proof** Can be found in [70].

# Chapter 3

# Related Work

Chapter 1 emphasized the need for pricing and provisioning of cloud resources. Cloud pricing and provisioning of resources have been receiving significant attention from both industry and academia. In this chapter, we present the review of state-of-the-art pricing and provisioning algorithms in the cloud. Initially, we analyze and categorize pricing algorithms depending on the knowledge about input - (i) algorithms with complete knowledge and (ii) algorithms with incomplete knowledge. Within each category, we discuss existing approaches in both cloud computing and algorithmic game theory. Afterward, we review existing cloud provisioning algorithms. Finally, we conclude the review by presenting a summary of the comparative analysis of the strengths and weaknesses of each existing pricing models.

## Contents

## 3.1 Pricing

Pricing strategies for resources in the cloud and edge environment are plentiful. We classify pricing approaches into two groups namely *offline* and *online* pricing based on the knowledge about input. In offline pricing, the algorithm has complete knowledge about inputs such as the number of buyers and utilities while inputs appear to online algorithms piece-by-piece.

### 3.1.1 Offline Pricing

Pricing decision involves service providers, customers, and resource demand. Conversely, pricing is a multiparticipant decision making. Generally, game theoretical models such as non-cooperative, Stackelberg models, etc., are applied in such a scenario. In cloud computing, the non-cooperative game theory is widely used for resource allocation [63]. In non-cooperative game theory, each player tries to maximize his or her payoff. However, non-cooperative game theory models assume complete information contrary to real systems. Alternative game theory models such as Stackelberg games are also explored. In Stackelberg game, one player acts as a leader and others are followers. Hence, alternatively called as the *leader-follower* game. Stackelberg games are applied in cloud computing as well for maximizing utilities [23] and guaranteeing QoS [79]. Wang *et al.* [80] propose an energy demand-based pricing for VMs based on the Stackelberg game to maximize profit. Stackelberg games require a leader and cannot be applied if players do not reach consensus on the leader.

In auctions, customers (or bidders) submit bids for items and auctioneer determine the winner based on his objective such as utility maximization. Landa *et al.* [7] perform Vickrey-auction based resource allocation, where each instance of a resource type by considering each instance of a resource type as an independent resource itself.

Gu *et al.* [81] propose a primal-dual based approximation algorithm for performing NFV service chain auctions. Buyers submit bids for the whole service chain. Unfortunately, their solution is also limited to service chains and not suitable in a market setting. Xilouris *et al.* [82] propose T-Nova, an auction-based marketplace for offering network functions to customers as-a-Service. T-Nova offers traditional auction mechanisms such as English, Dutch, and Vickrey. D'Oro *et al.* [83] implement service chain composition based on a marketplace approach. The servers behave as buyers and network request brokers act as customers to perform service composition at a minimum price.

Chichin *et al.* [35] propose a double-sided greedy auction algorithm for maximizing total utility of buyers. The algorithm performs the integral allocation. However, the algo-

rithm cannot handle resources with diminishing returns. Concretely, Jin *et al.* [6] propose a double-action based truthful bidding mechanism for resource sharing in for maximizing successful trading (number of matches between resources and customers). Here, buyers submit bids for the resources and the proposed mechanisms determine allocation and incentives. Pal *et al.* [24] propose a Walrasian market for cloudlets. They propose both a static and a dynamic version of the market with the goal of utility maximization for small cloud providers.

Market-based pricing is one of the significant areas in algorithmic game theory [69]. Specifically, Devanur *et al.* [27] propose a polynomial time algorithm for computing the market equilibrium based on the primal-dual and max-flow min-cut algorithms. Orlin [28] improves the overall computation time of market equilibrium by scaling [27]. Brânzei *et al.* [29] prove the existence of Nash equilibrium for constant elastic substitution utility and also bound the price of anarchy for Fisher markets. However, they also assume that resources are divisible. Cole and Vasilis [30] present an algorithm to compute market clearing prices and simultaneously perform integer allocation in a Fisher market. Anari *et al.* [34] extend the idea for multiple instances of resources using a stable polynomial method.

Goel and Vazirani [84] propose rational convex programming based approach to perform price discrimination of goods. Chakraborty *et al.* [36] propose a generic auction based algorithm for finding $(1+\varepsilon)$ market equilibrium for items with the limitation of fixed transaction costs.

### 3.1.2 Online Pricing

Cloud resource allocation is naturally online decision making [13, 45]. There are several works for determining cloud resource prices online. Zhang *et al.* [43] propose online auction mechanism for VMs based on the primal-dual framework to maximize the utility of both customers and service provider profit. Shi *et al.* [44] propose online combinatorial online auction for VMs. Furthermore, authors propose revenue maximizing online auction for virtual clusters [85]. The mechanisms presented satisfy individual rationality and incentive compatibility. Shi *et al.* [45] propose online combinatorial auction for heterogeneous VMs. Zhang *et al.* [86] propose an online auction-based marketplace for VNF service chains. The idea is to transform online stochastic social welfare to a deterministic fraction program based on bid arrival processes. The allocation is performed using the primal-dual method. Moreover, prices are learned based on historical bids, arrival, and departure of strategic bidders.

Zhao *et al.* [18] try to maximize profit for cloud service providers by joint modeling pricing, provisioning and job execution using stochastic optimization framework based on

Lyapunov optimization theory. Zhang *et al.* [86] propose an online stochastic auction mechanism for on-demand service chain provisioning and pricing at an NFV provider. Here, customers submit their bids and the algorithm computes both the allocation and the price for the service chains based on the primal-dual framework. The algorithm thus offers dynamic pricing and satisfies the individual rationality and incentive compatibility requirements. However, is further limited to VNFs, i.e., a single resource type and bids arrivals are assumed to be Poisson arrivals. Toosi *et al.* [42] propose online auction mechanism for cloud spot markets for maximizing profit.

Posted pricing is currently popular in online pricing market [63]. Zhang *et al.* [13] compute posted prices for cloud resources by designing exponential pricing function based on the primal-dual framework for 0-1 knapsack problem. Bubeck *et al.* [46] compute posted prices based on online learning approach for revenue maximization. Zhou *et al.* [87] design online auction for cloud jobs with deadlines. They maximize utility and compute posted prices using the primal-dual framework.

Online algorithms for market clearing prices are popular in algorithmic game theory. Angelopoulos *et al.* [47] propose deterministic and randomized algorithms for finding an approximate market equilibrium for a linear fisher market in an online setting where goods appear one-by-one. Blum *et al.* [88] propose a market clearing algorithm for the customer bids. The auctioneer has to decide whether to accept or decline a bid without knowledge of future bids. Bateni *et al.* [49] perform multiobjective optimization and dynamically allocate goods appearing online to budgeted buyers of a Fisher market with proportional fairness and efficiency as objectives. Azar *et al.* [48] design a primal-dual convex programming based algorithm for an online Fisher market where goods appear in each round with immutable decision.

## 3.2 Resource Provisioning

There have been several approaches that address the challenges of provisioning resources in the cloud (e.g., [51, 52, 55]). However, we believe that no previous work studied *what* resources should be provisioned in the context of complex services. The most similar work was recently published by Delimitrou *et al.* [89]. They proposed HCloud, a hybrid provisioning system that uses both reserved and on-demand resources (i.e., fits the business models of Amazon EC2). HCloud determines the best possible configuration by considering load-fluctuations, performance, and cost of resources. In contrast, this work focuses on describing a theoretical model based on robust queueing theory [70] that accurately describes the behavior of a given component in service regardless of the input traffic distribution. We are interested in understanding the exact behavior of each component in the service, not in

inferring it through micro-benchmarking as done by HCloud. Moreover, recently there have been some efforts on reducing the number of reconfigurations in cloud environments. For instance, Jiao *et al.* [90] studied the problem of resource allocation and reconfiguration in the multi-tier resource pool from an online optimization perspective. However, they do not address the issue of what configuration should be used.

Many queueing theory-based approaches have been proposed in the past. For instance, Urgaonkar *et al.* [58] models servers at each tier as a G/G/1 queue for representing arbitrary arrival and service time distributions. They assume uniform processing power and present both reactive and predictive provisioning algorithms. However, if the servers are allocated with different cores or processors, their model fails to capture this aspect due to the intractability of the G/G/m queue. Furthermore, Gandhi *et al.* [59] employed Kalman filters and queueing networks to scale a given application proactively. However, the biggest drawback is the assumption of Markovian arrivals and departures in a multi-tier architecture, contrary to reality [58]. Tsoumakos *et al.* [91] propose the TIRAMOLA framework for performing automatic resizing of NoSQL clusters based on a Markov Decision Process (MDP). The resizing operation is performed based on the user policies and current system state. Naskos *et al.* [92] extend the TIRAMOLA model to resizing clusters of a single generic application hosted on virtual machines. However, both approaches cannot be applied if a service comprises multiple *different* components.

Workflow deployment approaches are also widely used in the cloud [93–95]. This approach defines a workflow graph of services based on data dependencies. The workflow engine then scales the application based on input data and the workflow graph. In workflow based scheduling, the goal is to minimize the execution cost and to satisfy QoS [96, 97] by selecting an objective-maximizing path in a workflow graph in a *homogenous* environment. Similarly, Switch [98] is a recent project aiming to improve execution and deployment of time-critical applications, where the developers can specify QoS/QoE during the application's development.

## 3.3 Summary

Table 3.1 summarizes the recent offline pricing approaches in the cloud and edge computing. The state-of-the-art solutions feature a wide range of approaches that cause fractional allocation of resources except [35]. Additionally, these solutions do not strive to maximize NSW but usually aim for maximization of the profit at the providers' premises or utilitarian social welfare. Furthermore, we believe that none of these works considers *price discrimination*.

| Approach | Model | Maximization Objective | Domain |
|---|---|---|---|
| Daoud *et al.* [23] | Stackelberg game | Utilitarian | Cloud |
| Valerio *et al.* [79] | Stackelberg game | Profit | Cloud |
| Wang *et al.* [80] | Stackelberg game | Utilitarian | Cloud |
| Landa *et al.* [7] | Auction | Utilitarian | Cloud |
| Gu *et al.* [81] | Auction | Utilitarian | NFV |
| Xilouris *et al.* [82] | Auction | ★ | NFV |
| D'Oro *et al.* [83] | Auction | Utilitarian | NFV |
| Jin *et al.* [6] | Auction | Successful trades | Edge |
| Chichin *et al.* [35] | Auction | Utilitarian | Cloud |
| Pal *et al.* [24] | Walrasian | Utilitarian | Edge |

Table 3.1: A summary of related works with respect to offline pricing

★ feature not specified

| Approach | Model | Maximization Objective | Domain |
|---|---|---|---|
| Zhang *et al.* [43] | Auction | Utilitarian | NFV |
| Shi *et al.* [44] | Auction | Utilitarian | Cloud (VMs) |
| Shi *et al.* [85] | Auction | Utilitarian | Cloud (Virtual clusters) |
| Shi *et al.* [45] | Auction | Utilitarian | Cloud (heterogeneous VMs) |
| Zhang *et al.* [86] | Auction | Utilitarian | NFV |
| Toosi *et al.* [42] | Auction | Profit | Cloud |
| Zhao *et al.* [18] | Stochastic optimization | Profit | Cloud |
| Zhang *et al.* [13] | Posted pricing | Utilitarian | Cloud |
| Zhou *et al.* [87] | Posted pricing | Utilitarian | Cloud |
| Bubeck *et al.* [46] | Posted pricing | Profit | Cloud |

Table 3.2: A summary of related works with respect to online pricing

Similarly, Table 3.2 summarizes recent online pricing approaches in cloud computing. The state-of-the-art online pricing approaches in the cloud and edge computing are concerned mainly utilitarian. There are approaches in algorithmic game theory for maximizing NSW [48,49]. However, [48] cannot be applied for varying customers and integer allocation cannot be guaranteed by [49].

There have been several approaches that address the challenges of provisioning resources in the cloud [51, 52, 55, 89, 90]. Many queueing theory-based approaches have been proposed in the past [58, 59]. There are works on resizing clusters [91, 92]. Further, workflow deployment [93–95] and scheduling [96, 97] approaches are also widely used in the cloud. However, we believe that no previous work studied *what* resources should be provisioned in the context of complex services.

# Chapter 4

## ARPP Sealed-bid Auction

Auctioning is one of the popular forms of dynamic pricing model that helps to find a fair price during price uncertainty [38] without compromising economic efficiency [37]. As we see in chapter 3, a majority of sealed auction algorithm at least assume prior knowledge about bid distributions. This chapter presents an *incentive compatible* auction mechanism dubbed RAERA for auctioning edge resources.

Initially, we motivate and formalize the problem of designing auction mechanism for the *edge*. Subsequently, we propose RAERA algorithm. Finally, we evaluate and present experimental results.

### Contents

## 4.1 Introduction

The Internet is one of the vital infrastructures of our society. With the advent of the Internet of Things (IoT) and Machine-to-Machine communication, it is expected to connect approximately 50 billion devices by 2020 [99]. To cope with this growth Internet stakeholders have recently deployed new technologies in both access and core networks. In particular, infrastructure and service providers currently moving towards edge computing, where data, applications, and services are placed towards the edge of the network rather than in centralized locations. The benefits of this strategy are [4–7]

(i) Lowering the traffic passing through the infrastructure.

(ii) Reducing the latency for services.

(iii) Scaling network services.

By employing edge computing, infrastructure providers (sellers) are also gaining a new set of customers. Content and service providers are looking at means to enhance their users' experience to gain a competitive advantage by pushing content and services closer to the edge and thus closer to the users. Currently, these customers (buyers) can buy computing and network resources at the cloud in services such as Amazon CloudFront and Microsoft Azure. Here, a buyer typically visits the seller's marketplace, manually selects resources based on his requirements and subsequently pays for their usage time. The predominant pricing model for the instances is *fixed pricing*, i.e., a particular instance has the same price to all users at all times. A variant of this is the *differential pricing* where specific buyers get discounted rates (similar to frequent flyer programs) or the rates could be different during specific time periods (e.g., weekend tariffs).

In these pricing models, it is difficult to arrive at a fair price[1] for all participants [38]. In markets that exhibit uncertainty about the price of goods or services and that have little knowledge about market participants, there is a tendency for the sellers to set a price that is favorable to them [15]. However, in the presence of competition, these pricing schemes could either drive customers away or result in lower prices leading to a loss. Therefore, some sellers opt for *dynamic pricing*, in which prices fluctuate based on demand and usage patterns. For instance, Amazon offers spot instances, where a buyer specifies the maximum amount for a particular instance. The allocated instance is terminated when the bid amount exceeds the current spot instance price. Price uncertainty is the major obstacle for *dynamic pricing*.

Auctioning is a form of dynamic pricing that helps to find a fair price in cases where

[1] with respect to terms and conditions

both the seller(auctioneer) and the buyer do not know the true value and estimates regarding the terms and conditions are highly imperfect (price uncertainty) [38]. Here, buyers (or bidders) submit bids for items and usually the bidder with the highest bid is the winner. Most of the auction mechanism assume prior distribution. However, if a user draws a bid from a different probability distribution, then the obtained price is not optimal [40,41] which is often the case in edge computing, where each user is different and may draw valuations from a different probability distribution.

There are several auction-based resource allocation works in the Edge [6, 7]. However, these solutions lack a generic solution to deal with heterogeneous resources and thus with market uncertainty. In this chapter, we propose the first auction mechanism based on robust optimization for the edge computing scenario. We develop our approach based on robust optimization for performing multi-item auctions [40]. Our mechanism—dubbed *Robust Auction for Edge Resource Allocation* (`RAERA`) —creates an uncertainty set for each resource based on the historical bidding data maintained by the infrastructure provider, and can thus deal with the unknown market conditions. In the first stage, `RAERA` computes a *reservation price* for each resource based on buyers' historical data. The reservation price is the minimum price that an infrastructure provider needs to sell for avoiding loss. In the second stage, `RAERA` allocates the resources for which bid value is higher than its reserve price.

Finally, it computes payment for each bidder. Importantly, `RAERA` offers the property of *individual rationality* (bidders do not derive negative utility for truthful bids) and is *incentive compatible* (truthful bids have higher utility for the bidder compared to non-truthful bids). Our evaluations highlight that `RAERA` offers a time-dependent fair price. Sellers can achieve higher revenue in the range of $5\% - 15\%$ irrespective of varying demands, and the buyers pay up to 20% lower than their top bid amount.

## 4.2 `RAERA` Problem

This chapter addresses the problem of auctioning edge resources to achieve both profit and social welfare for the resource seller(s) and the resource buyers. Furthermore, this chapter defines these terms as follows:

- **(Edge) Resources** are resources that could be used to support edge computing. These resources are on the Edge in various forms such as micro data centers, cloudlets, and fog nodes. In this chapter, resources are assumed to be atomic *units* (I.e., the smallest unit of resource that could be made available) that could be, for example, a) virtualized resources such as VMs, dockers, and unikernels; b) actual resources

such as CPU cores (or fractions thereof) and memory; or c) indirect resources such as execution time.

- **Sellers** are those entities who own edge resources at the edge of a network and are willing to lease them to interested buyers. For example, infrastructure providers such as edge/access/eye-ball ISPs and mobile network providers can be sellers.
- **Buyers** are those who would like to deploy their services on edge resources to enhance the user experience. For example, content providers or service providers can be buyers.

Here, dynamic pricing leads to cheaper resources for the buyer and better utilization for the seller [63]. However, it is necessary to address the following challenges:

- **Uncertainty about pricing**: Due to a lack of ground truth, it is difficult to validate resource valuations. For instance, it is difficult not only to infer but also to validate CPU consumption of multiple VMs sharing the same physical core. Hence, pricing these resources is non-trivial, and sellers are uncertain about the resource prices they should call for. Shi *et al.* [4] emphasize the need for a new cost model, especially for edge resources, which not only guarantee profitability for sellers but also acceptability from buyers.
- **Service guarantee**: Currently, service providers cannot guarantee service availability even with fixed pricing [61]. For instance, allocated amazon spot instances are terminated as soon as the current called-for price exceeds the buyer's bid. As a result, a buyer has to bid higher to ensure instance availability for critical applications. Otherwise, procure instances with fixed pricing scheme, which favors the seller of the resources.
- **Strategic behavior**: Buyers often also have strict requirements on the resources (e.g., if they need to meet SLOs), but most of the sellers are unable to provide service guarantees within fixed pricing for premium customers [61]. As a result, buyers usually try to receive better service than required by inflating their requirements, incurring unnecessary costs at the sellers' premises [61].

This chapter addresses these challenges by following a sealed bid auctioning approach. Here, the buyers submit sealed bids for resources and receive resources only if they are winners in the auction. Specifically, auctions can counter the price uncertainty problem, provide service guarantees since items are entirely allocated to the winner [21], and *incentive compatible* auctions encourage truthful bidding. One popular example is the *Vickrey auction*, where the bidder with the highest bid wins the auction but only pays the second highest bid instead of his quoted bid, which has proven to prevent strategic behavior of buyers [100].

In this subsection, we formally define our problem as follows: Let $N = \{1,2,\ldots,n\}$ be a set of $n$ buyers indexed by $i$, i.e., $i$ represents $i^{th}$ buyer. Similarly, let $M = \{1,2,\ldots,m\}$

be the set of edge resources indexed by $j$, i.e., $j$ represents the $j^{th}$ resource. Each buyer $i$ has a private valuation for each resource. In our model, $v_{ij}$ represents the valuation of buyer $i$ for the $j^{th}$ resource. The valuation profile or vector of a buyer $i$ is represented by $v_i = (v_{i1}, v_{i2}, \ldots, v_{im})$. Each buyer $i$ submits a bid represented by $b_i = (b_{i1}, \ldots, b_{im})$ for $m$ resources. Let $\beta \in \mathscr{R}^n$ be the bid profile or vector of all the users, i.e., $\beta = (b_1, b_2, \ldots, b_n)$. Let $b_{-i}$ be the bid vector of all users except $i$, i.e., $b_{-i} = (b_1, b_2, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$. Also, $\beta = (b_i, b_{-i})$.

Let $x(\cdot) : R^n \rightarrow N$ be the allocation function of edge resources to users based on their submitted bids.

If $v$ is a bid profile of successful resource allocation, then $x(v) = \{x(v)_{ij}\}, \forall i \in N, j \in M$ where $x(v)_{ij}$ is the amount of resource $j$ a bidder $i$ receives when the profile is $v$. In our model, $x(v)_{ij}$ is discrete to represent an atomic resource, i.e., the allocation $x(v)_{ij}$ is an integer since a fractional value would represent a fractional unit and thus does not have any practical significance.

Similarly, let $p(\cdot) : R^n \rightarrow R$ be the payment function which maps a bid profile to an n-dimensional real value which is a monetary payment, i.e., $p(v) = (p_1(v), \ldots, p_n(v)$. Informally, $p_i(v)$ is the monetary payment of buyer $i$ to the seller.

Let $u_i$ be the utility derived by the $i^{th}$ buyer. Informally, a utility is the satisfaction obtained by consuming goods. We assume the utility $u_i$ is quasilinear. Quasilinearity implies the risk neutrality for buyers [62], i.e., the more money a buyer spends, the greater the utility. Formally, $u_i(v_i, x(b), p(b)) = v_i x_i(b) - p_i(b)$.

Let $U_j \in \mathscr{R}^n$ be the uncertainty set (i.e., data and constraints are allowed to change within the set without affecting optimality) for the $j^{th}$ resource. Bid vectors for the $j^{th}$ resource are drawn from this uncertainty set. The uncertainty set for an auction is constructed based on historical bid data. According to the central limit theorem, a distribution with a mean $\mu$ and variance $\sigma^2$ will converge to standard normal distribution when the sample size $n \rightarrow \infty$. Based on [101], we define the uncertainty set $U_j$ for the $j^{th}$ resource as follows:

$$U_j = \left\{ (b_{1j}, \ldots, b_{nj}) \middle| -\Gamma \leq \frac{\sum_{i=k+1}^{n} b_{kj} - (n-k)\mu_j}{\sqrt{(n-k)}.\sigma_j} \leq \Gamma, k \neq n \right\} \tag{4.2.1}$$

For a standard normal distribution $\mathbb{Z}$, the confidence interval is 95% and 99% for $|Z| \leq 2$ and $|Z| \leq 3$ respectively, i.e., $\mathbb{P}(|Z| \leq 2) \approx 0.95$ and $\mathbb{P}(|Z| \leq 3) \approx 0.99$. Hence, in Eq. (4.2.1), $\Gamma$ is chosen between 2 and 3 to reflect the desired confidence interval. Also, $(n-k)$ can be regarded as the window of recent historical values. If $k = 0$, then all the historical values are considered. Furthermore, $\mathscr{U}$ denotes the uncertainty set for $m$ resources, i.e.,

$\mathscr{U} = U_1 \times \cdots \times U_m$.

In our model, all bids and valuation vectors are drawn from the uncertainty set $\mathscr{U}$, i.e., $\forall i \in N, b_i, v_i \in \mathscr{U}$. Let $\mathscr{W}$ be the *break-even optimal* (optimal revenue in presence of total uncertainty [41]) of the service provider. The goal of this chapter is to maximize $\mathscr{W}$ satisfying following properties:

- **Individual Rationality (IR):** The users should not lose money when they bid truthfully, i.e., $u_i \geq 0$.
- **Incentive compatibility (IC):** The user has to derive higher utility for truthful bidding. Let $\widehat{b_i}$ be a non-truthful bid profile and $x(\widehat{b_i})$ and $p(\widehat{b_i})$ corresponding allocation and payment. Then, this property implies that $u_i(v_i, x(b_i, b_{-i}), p(b_i, b_{-i})) \geq u_i(v_i, x(\widehat{b_i}, b_{-i}), p(\widehat{b_i}, b_{-i}))$.

We formulate our goal as the following linear optimization problem.

$$
\begin{aligned}
\underset{x}{\text{maximize}} \quad & \mathscr{W} \\
\text{subject to} \quad & (i) \quad \mathscr{W} - \sum_{i \in N} p_i \leq 0, \\
& (ii) \quad \sum_{j \in M} x_{ij} \leq 1, \quad \forall i \in N, \\
& (iii) \quad \sum_{k=1}^{m} \widehat{b_{ij}} x_{ij} - \widehat{p_i} \leq \sum_{k=1}^{m} b_{ij} x_{ij} - p_i, \quad \forall b_{ij}, \widehat{b_{ij}} \in \mathscr{U}, \\
& (iv) \quad p_i \leq B_i, \quad \forall i \in N, \\
& (v) \quad p_i \leq \sum_{j \in M} b_{ij} x_{ij}, \quad \forall i \in N, \\
& \quad x_{ij} = \{0, 1\}, \quad \forall i \in N, j \in M.
\end{aligned}
\tag{4.2.2}
$$

Here, constraint (i) ensures a non-negative profit for the auctioneer, (ii) implies allocation of every resource, (iii) and (iv) ensure the IC and IR property, respectively; (v) implies the atomic resource allocation to $i^{th}$ buyer. Furthermore, the notations are summarized in Table 4.1.

## 4.3 RAERA Algorithm

In this section, we present the algorithm for the optimization problem defined in Eq. (4.2.2). The goal of Eq. (4.2.2) is to maximize the worst-case revenue $\mathscr{W}$ for the auctioneer—this is the *least upper bound* or *supremum* revenue irrespective of the probability distribution of

| Symbol | Description |
|---|---|
| $N$ | Set of buyers |
| $M$ | Set of edge resources |
| $\mathscr{W}$ | Break-even optimal revenue |
| $b_i$ | True bid of buyer $i$ |
| $\widehat{b_i}$ | Reported bid of buyer $i$ |
| $\beta$ | Bid profile of all buyers |
| $x(\cdot)$ | Allocation function |
| $\nu$ | Bid profile of successful allocation |
| $p(\cdot)$ | Payment function |
| $u_i$ | Utility of buyer $i$ |
| $v_i$ | Valuation of buyer $i$ |
| $p_i$ | Payment received by buyer $i$ |
| $U_j$ | Uncertainty set of edge resource $j$ |
| $\mu_j$ | Mean of edge resource $j$ historical bids |
| $\sigma_j$ | Variance of edge resource $j$ historical bids |
| $\Gamma$ | Variability parameter |

Table 4.1: A summary of notations used for proposing RAERA.

buyers bids. Here, it is crucial for the seller to determine a minimum price for the resources, which guarantees non-negative profit and is popularly known as the *reserve price* [21].

The seller will have information about historical bid valuations only, i.e., the uncertainty set $\mathscr{U}$. Let $r_{ij}$ be the reserve price of resource $j$ for the bidder $i$. The reserve price implies that bidder $i$ should pay at least $r_{ij}$ for getting a non-zero allocation of the $j^{th}$ resource. We can have multiple valuations for the items in the uncertainty set and have to find the valuation which will maximize the revenue for the auctioneer. The auctioneer should calculate the reserve price for resources to maximize the revenue and to avoid financial loss in case of selling below the reserve price.

The idea is to apply duality theory. Section 2.1.1 presents the basics of duality theory. The dual forms a lattice and a corner of the lattice represents a Vickrey pay-off point where the buyers have an incentive for their truthfulness in a sealed bid auction [102]. In our case, finding the *break-even optimal* valuation is the primal problem. Formally, let $\boldsymbol{z} = \{z_{ij}\}, \forall i \in N, j \in M$ denote the worst case valuation vector. We solve the Eq. (4.3.1) to find $\boldsymbol{z}$ for all the valuations $v$ in uncertainty set. The corresponding allocation also called as candidate allocation (allocations before receiving the bid vector) is represented as $x^{\star}_{ij}$,

$$
\begin{aligned}
\underset{v \in \mathscr{U}}{\text{maximize}} \quad & \sum_{i \in N} \sum_{j \in M} x_{ij} \cdot v_{ij} \\
\text{subject to} \quad & (i) \quad \sum_{j \in M} x_{ij} \leq 1, \quad \forall i \in N, \\
& (ii) \quad \sum_{j \in M} x_{ij} \cdot v_{ij} \leq \sum_{j \in M} x_{ij} \cdot u_{ij}, \quad \forall i \in N, u_{ij} \in \mathscr{U}, \\
& (iii) \quad x_{ij} \geq 0, \quad \forall i \in N, j \in M.
\end{aligned}
\tag{4.3.1}
$$

In Eq. (4.3.1), constraint (i) ensures the allocation of every resource in the auction, (ii) is a robust optimization constraint for polyhedral uncertainty that ensures the IC and IR properties [40], and (iii) ensures a zero or non-negative allocation (a zero allocation implies that the bidder is not the winner).

Then, the dual of Eq. (4.3.1) is as follows:

$$
\begin{aligned}
\underset{\varepsilon, \eta}{\text{minimize}} \quad & \sum_{j \in M} \varepsilon_j + \eta_i \sum_{i \in N} \sum_{j \in M} x^{\star}_{ij} \cdot \tilde{u}_i \\
\text{subject to} \quad & \varepsilon_j + z_{ij} \cdot \eta_i \geq z_{ij}, \quad \forall i \in N, j \in M, \\
& \varepsilon_j, \eta_i \geq 0.
\end{aligned}
\tag{4.3.2}
$$

The dual value for the uncertainty is denoted as $\tilde{u}_i$ and calculated as follows:

$$\tilde{u}_i = \arg\min \sum_{j \in M} x^{\star}_{ij} \cdot u_{ij}, \forall i \in N \tag{4.3.3}$$

The reserve price $r_{ij}$ is calculated as follows:

$$r_{ij} \leftarrow \varepsilon^{\star}_j + \eta^{\star}_i \cdot x^{\star}_{ij}, \forall i \in N, j \in M \tag{4.3.4}$$

Algorithm 4.1 presents the pseudo-code for calculating reserve price from the uncertainty set.

**Algorithm 4.1** calculateReservePrice

1: **procedure** CALCULATERESERVEPRICE($\mathscr{U}$)
2: Compute worst case valuation $z = \{z_{ij}\}$ and candidate allocation $x^{\star} = \{x^{\star}_{ij}\}$ using Eq. (4.3.1)
3: Calculate $\varepsilon^{\star}$ and $\eta^{\star}$ using Eq. (4.3.2)
4: Calculate $r_{ij}$ using Eq. (4.3.4)
**return** $r_{ij}, x^{\star}_{ij}, \forall i \in, j \in M$

Once the reserve price is determined, the subsequent step is to filter bids less than reserve price and such bids are knowns as *feasible bids*. Let $\mathscr{P}$ be the set of *feasible bids* issued by all buyers and defined as follows:

$$\mathscr{P} = \arg \left\{ \begin{array}{l} \sum_{i \in N} y_{ij} \leq 1 - \sum_{i \in N} x^{\star}_{ij}, \forall j \in M \\ \sum_{j \in M} y_{ij} \cdot u_{ij} - \sum_{j \in M} x^{\star}_{ij} \cdot r_{ij} \\ + \sum_{j \in M} x^{\star}_{kj} \eta_i \tilde{u}_j, \forall u \in U, \forall i \in N \end{array} \right\} \tag{4.3.5}$$

Next, we find an allocation in $\mathscr{P}$ which maximizes the revenue for the auctioneer. Let $y = \{y_{ij}\}$ be the allocation. Formally,

$$\{y_{ij}\} = \arg\max_{\mathscr{P}} \sum_{i \in N} \sum_{j \in M} y_{ij} \cdot (b_{ij} - r_{ij}) \tag{4.3.6}$$

We adjust candidate allocation to reflect input bids for every buyer $i$. Let $a_{ij}$ is the final allocation and computed as:

$$a_{ij} = x^{\star}_{ij} + y_{ij}, \forall i \in N, j \in M \tag{4.3.7}$$

The payment is computed based on both the reserve price and the impact of buyers valuation. Moreover, the primary goal of payment is to induce truthfulness during bidding.

Hence, we determine allocation without each buyer. Let $\mathscr{Q}_{\|}$ be the set of allocations when buyer $k$ is absent. It is determined as follows:

$$\mathscr{Q}_k = \arg \left\{ \begin{array}{l} \sum\limits_{i \in N} y_{ij} \leq 1 - \sum\limits_{i \in N} x^{\star}_{ij}, \forall j \in M \\ \sum\limits_{j \in M} y_{ij} \cdot u_{ij} - \sum\limits_{j \in M} y_{ij} \cdot r_{ij} \forall u \in U, \forall i \in N \backslash \{k\} \end{array} \right\} \tag{4.3.8}$$

Let $y^{-k} = \{y^{-k}_{ij}\}, i \in N \backslash k, j \in M$ be the allocation for set $\mathscr{Q}_k$. We have,

$$\{y^{-k}_{ij}\} = \arg\max_{\mathscr{Q}_k} \sum_{i \in N \backslash \{k\}} \sum_{j \in M} y_{ij} \cdot (b_{ij} - r_{ij}) \tag{4.3.9}$$

Finally, we compute the payment for each buyer as follows:

$$\begin{aligned} p_k = \sum_{j \in M} y_{kj} \cdot r_{kj} + \sum_{j \in M} x^{\star}_{kj} \cdot r_{kj} - \sum_{j \in M} x^{\star}_{kj} \cdot \eta^{\star}_i \cdot \tilde{u}^k_j \\ + \sum_{i \in N \backslash \{k\}} \sum_{j \in M} y^{-k}_{ij} \cdot (b_{ij} - r_{ij}) \\ - \sum_{i \in N \backslash \{k\}} \sum_{j \in M} y_{ij} \cdot (b_{ij} - r_{ij}), \forall i \in N \end{aligned} \tag{4.3.10}$$

In Eq. (4.3.10), $p_k$ represents the amount a buyer has to pay to be part of successful allocation. The fourth term represents the profit when buyer $k$'s valuation is absent while the fifth term computes profit when valuation is present. The payment rule presented satisfies the IC and IR properties (theoretical proof can be found in [40]). We present Algorithm 4.2 for edge resource allocation.

**Algorithm 4.2** calculateReservePrice

**procedure** ALLOCATEANDPAY($b, r_{ij}, x^{\star}_{ij}$)
  **if** $b \notin \mathscr{U}$ **then**
    $a_{ij} \leftarrow 0, p_i \leftarrow 0, \forall i \in N, j \in M$
  Compute set $\mathscr{P}$ using Eq. (4.3.5)
  Compute set $\mathscr{Q}_k$ using Eq. (4.3.8) $\forall k = 1, \ldots, n$
  Compute $y_{ij}, \forall i \in N, j \in M$ using Eq. (4.3.6)
  $a_{ij} \leftarrow x^{\star}_{ij} + y_{ij}, \forall i \in N, j \in M$
  Compute payment $p_i$ using Eq. (4.3.10)
  Allocate $j^{th}$ resource to $i^{th}$ bidder with a probability $a_{ij}$
  Compute price for $j^{th}$ resource for $i^{th}$ bidder as $\frac{p_i}{\sum_{j \in M} a_{ij}}$ **return** $a_{ij}, p_i, \forall i \in, j \in M$

Figure 4.1 shows the RAERA auction framework. The bidders submit the sealed bid for the resources. The service provider computes the reserve prices and candidate using algorithm 4.1 from the uncertainty set constructed from the previous bid values. Once, reserve

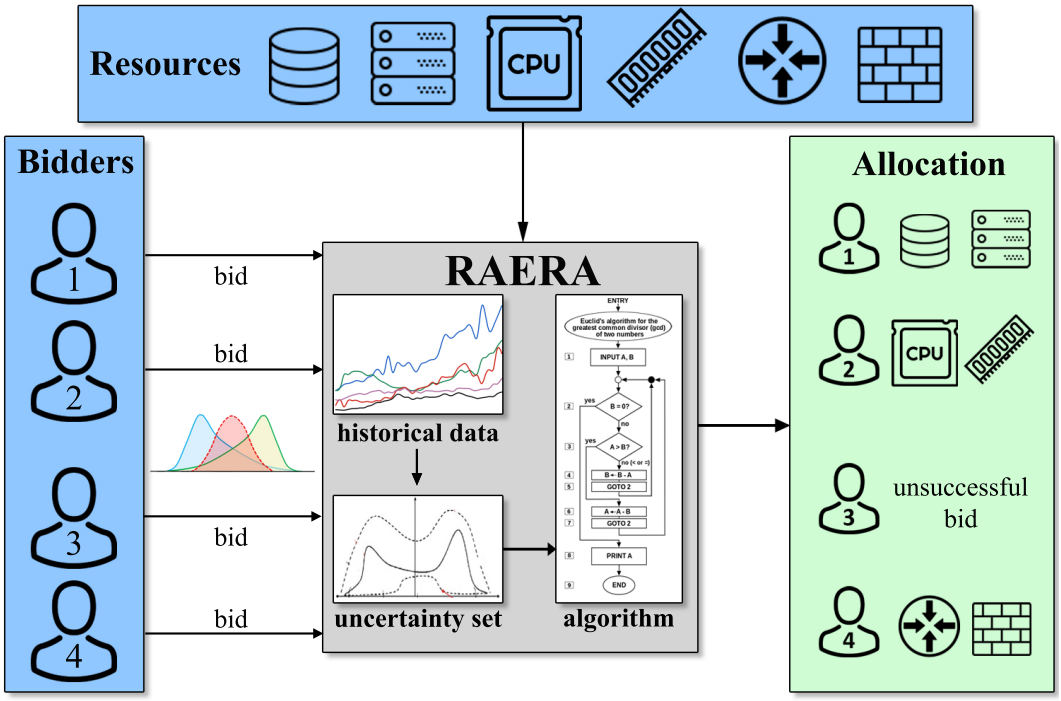



Figure 4.1: RAERA Auction Framework: Buyers place bids on resources, and RAERA decides the allocation.

prices are computed final allocations, and prices are determined using Algorithm 4.2. Since RAERA is incentive compatible, the bidders have to bid truthfully. Otherwise, it will affect their incentives.

## 4.4 RAERA Evaluation

We now evaluate RAERA by simulation to study its benefit for both sellers (increased profit) and buyers (fair price).

### 4.4.1 Methodology

RAERA requires knowledge of historical bids to create its uncertainty sets. This data is private to infrastructure providers. Further, we believe that a real edge resource demand dataset is unavailable publicly. Moreover, edge resource providers don't reveal real information about pricing publicly due to fear of losing competitive advantages.

We, therefore, model network traffic (and subsequently resource demand) based on Gill *et al.* [103], who provide insights into YouTube traffic demand on an hourly scale. Gill *et al.* [103] is the first work to analyze and characterize Youtube traffic to understand how user contents are viewed and distributed over the internet based on YouTube data collected for three months. The collected data is a combination of both local (on campus) and global internet. They observed a steady rise in traffic during the morning with a peak achieved during the afternoon. For instance, YouTube traffic reaches a peak during the afternoon, while only 5% of peak traffic is present in the early morning. Based on these trends we generate bid valuations for every hour: buyers bid higher in the afternoon compared to morning hours.

We generate both historical data and bid vectors using a uniform distribution. The bid vector is 20% higher compared to historical data. We use CPLEX to solve the optimization problem formulated in Algorithms 4.1 and 4.2. Finally, our simulator calculates the payment for each buyer in case of a successful allocation.

### 4.4.2 Results

We start by investigating seller profit. Figure 4.2 compares the guaranteed profit of the seller with the profit achieved by RAERA. The guaranteed profit is what a seller will earn regardless of the bid probability distribution. We observe that RAERA can adapt resource prices based

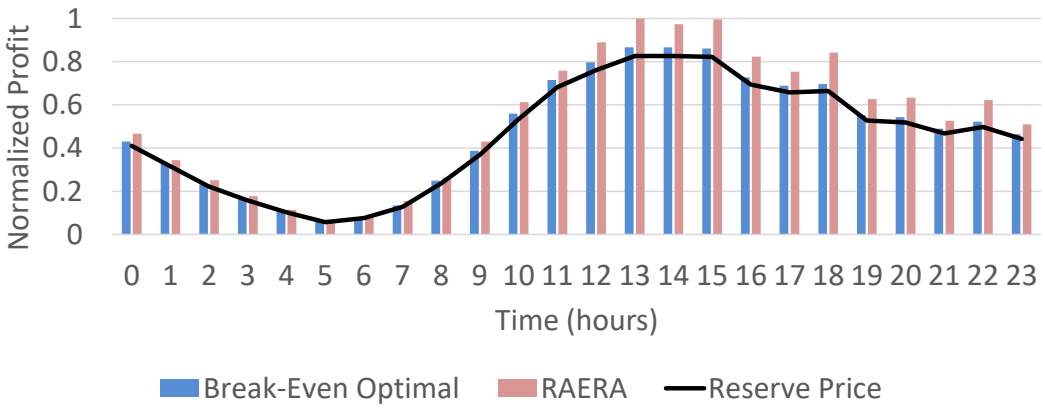



Figure 4.2: Profit: Break-even Optimum vs RAERA

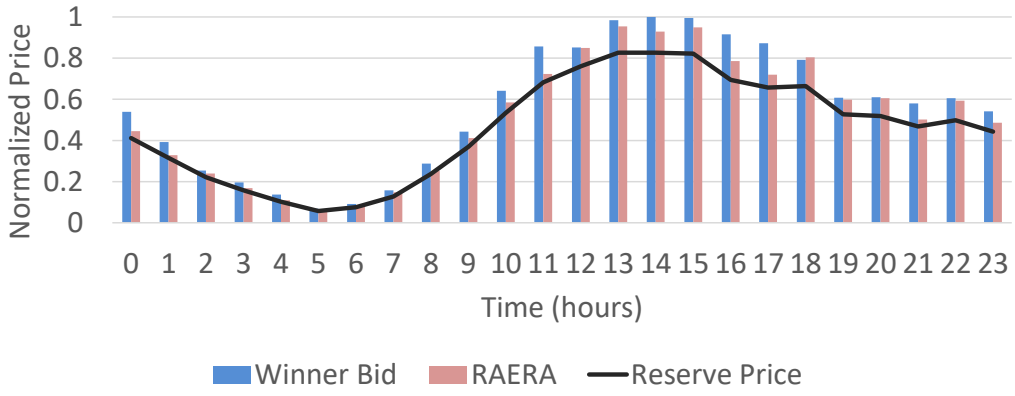



Figure 4.3: Price: Average Bid vs RAERA

on demand and achieves a 5% to 15% higher revenue irrespective of demand even during low demand periods. Further, RAERA is beneficial to the buyers as well. According to Figure 4.3, the bids of buyers can be up to 20% lower using RAERA. Lower bids on average also result in lower payment on average.

## 4.5 Summary

In this chapter, we introduced RAERA, a novel method based on robust optimization for multi-item auctions for edge computing resources. In preliminary experiments, we have shown that RAERA can guarantee the profit even in a scenario loaded with uncertainty, unlike traditional methods. Furthermore, it is incentive compatible and satisfies individual rationality. Therefore, RAERA can set a time dependent fair price that benefits both buyers and sellers.

# Chapter 5

# ARPP Market

In *differential pricing*, each customer pays different prices for the same item. Consequently, this generates additional revenue for producers. Generally, cloud customers are ready to pay an additional price for services with higher QoS. Furthermore, differential pricing can influence customer spending. For instance, green energy-based VMs can be priced lower than their brown energy counterpart to increase green energy adoption. This chapter proposes an auction-based Edge resource market (ERM) for computing differential prices in the Fisher market.

Initially, we motivate the need for NSW maximization and *differential pricing* in the *Cloud* and *Edge*. Afterward, we formalize the problem of implementing of *differential pricing* in the Fisher market. Subsequently, we propose ERM algorithm. Finally, we evaluate and present experimental results.

## Contents

## 5.1 Introduction

Edge computing is gaining in popularity for a variety of applications. For instance, edge computing can support the trend to virtualize network functions [104] by i) providing these virtualized services close to the user [105, 106]; ii) avoiding the need to redirect traffic to cloud services [107], and iii) supporting the scaling up/down of services to meet user demand [108]. Further, IoT applications such as those on vehicles, smart-homes, and factories could benefit from edge computing [109] to avail virtualized services such as pre-processing a significant amount of data collected. Edge-computing could also help alleviate the latency concerns of making use of cloud services for computation offloading [4] by providing the extra computation resources close to the users [4–7]. In this context, a market provider typically provisions requested edge resources to the customer (e.g., via Amazon CloudFront or Amazon EC2).

One key consideration of the provider in the request-to-provision process is *pricing*. However, pricing is non-trivial in cloud and edge due to following reasons:

(C1) High prices may drive away customers while low prices may result in reduced profit.

(C2) Resources in edge or cloud data centers are generally virtualized. While the cost of physical resources (e.g., the cost of buying and running a physical server) is tangible, this is not the case for virtual resources. Hence, determining actual resource usage is uncertain [18].

(C3) Determining operating cost is non-trivial since multiple virtual machines (VMs) usually share the same physical resources in a highly dynamic environment, and it is unclear how many resources the provisioning of a particular VM will consume.

Therefore, both academia and industry [17, 18, 80, 110–112] frequently propose pricing strategies for the cloud.

Currently, cloud and edge providers follow a "pay-as-you-go" pricing model. In this model, prices are *fixed* and charged based on usage. For instance, Amazon EC2 instances are charged hourly. Moreover, fixed pricing fails to capture resource supply and demand [113]. Consequently, providers suffer losses for scare edge resources(high demand) as these are at the same price as the other provisioned resources (low demand) in a cloud data center. Most often, fixed pricing favors providers contractually [19]. Finally, in the presence of only a few big cloud service providers (e.g., Amazon or Microsoft) cloud resource prices are *oligopolistic* [16, 17]. Oligopolistic prices are higher than competitive prices, and, if more resource providers would enter the market, the market is poised to become a low commodity market with low-profit margins [16].

One option to appropriately set prices based on supply and demand is *dynamic pricing*.

Dynamic pricing maximizes the provider's profit and at the same time has the potential to be fair to all users [18]. However, dynamic pricing is not trivial, especially in edge data centers, where resource choices are plentiful and highly variable, the associated costs such as energy costs could vary based on location, and the actual resource usage is uncertain [18]. In particular, the following properties of pricing are desired from the provider's point of view.

(i) Dynamic pricing should determine the optimal price such that there is at least one buyer who is interested in each resource, based on supply and demand—i.e., the pricing scheme satisfies the **market clearing** property [21] which guarantee complete allocation of all the available resources. This pricing is also known as *equilibrium* pricing.

(ii) The Market provider will strive for increasing the revenue acquired from selling the resources.

(iii) **Differential pricing** among resources should be possible for multiple reasons. For instance, the same VM can be more costly when provisioned during peak times (when resources are more scarce) than when provisioned during off-peak hours. Further, running on servers fueled by brown energy should be priced differently than resources running on green energy, and a VM provisioned on the network edge should be more costly than the same VM provisioned at a cloud data center. Also, cloud resources often suffer from diminishing returns, i.e., often adding more copies of the same resource does not improve the utility for the buyer [31].

State-of-the-art solutions currently neither offer capabilities for differential pricing [6,24] nor consider diminishing returns. In algorithmic game theory, the *Separable Piecewise-Linear Concave* (SPLC) utility models diminishing returns [33]. Previous works extend max-flow min-cut market algorithms for the SPLC utility for multiple copies and also propose stable polynomial based rounding [34]. However, these solutions are computationally complex and a stable polynomial approach requires highly complex ellipsoidal algorithms.

Additional existing works typically also assume that resources are divisible and thus perform fractional allocation of resources to buyers [26, 28, 114, 115]. However, many cloud resources (e.g., VMs) cannot be allocated fractionally, which is usually solved by rounding fractional allocations to the nearest integer solution. Unfortunately, the resulting rounding difference (*integrality gap*) is unbounded for a market, i.e., the difference between optimal objective value and rounded objective value grows with the number of buyers and resources [30, 34].

To overcome these issues, and to efficiently arrive at a market clearing price while ensuring that price discrimination reflects supply and demand, we propose the Edge Resource Market (ERM), a market-based pricing scheme predominantly for edge computing resources

based on auctions. We believe that edge computing is an ideal use-case since—similar to how individuals could contribute electricity to the grid and earn money [116, 117]—such a market mechanism could encourage a large number of smaller players to participate as either sellers or buyers of such resources since otherwise the cloud market is poised to become a low commodity market [16].

In ERM, we assume the presence of multiple buyers and multiple resources types. Each resource type can have multiple instances, and each instance of one resource type has the same functionality but different specification. Hence, each of the buyers can have a different utility (or valuation) for (a subset of) the resource instances. Our goal is to maximize the social welfare of buyers and customers. In cloud computing, maximizing social welfare improves overall system efficiency and customer experience simultaneously [13]. There are three types of social welfare namely *utilitarian*, *egalitarian* and **Nash Social Welfare (NSW)**.

The goal of *utilitarian* solutions is to maximize the total utility across the buyers. Hence, the allocation is thus always biased towards the buyers with higher utility. On the other hand, the *egalitarian* or *max-min fairness* approach tries to maximize the happiness of the least satisfied buyer and is thus biased towards buyers with the lowest utility. NSW is a Pareto outcome between *utilitarian* and *egalitarian* approaches [25, 30]. In other words, NSW achieves the balance between *efficiency* and *fairness*. Hence, in this work, our goal is to maximize NSW.

A market which maximizes NSW is the well-known Eisenberg-Gale or Fisher market [26]. The prices in the Fisher market is *market clearing* [26]. Further, prices are determined solely on supply and demand which is determined from the customer utility. Hence, there is no need to measure actual consumption of resources for computing prices. Therefore, challenges C2 and C3 are not applicable for the Fisher market.

Based on these principles, our contributions are:

- We propose ERM, an auction-based algorithm (Section 5.3) for computing the $(1+\varepsilon)$ equilibrium for markets with buyers with SPLC utility and market providers that offer multiple resource instances. Typically, prices in an approximate market equilibrium are market clearing, and most buyers achieve a satisfactory allocation [118]. The basic idea is to allocate resource instances requested by a set of buyers by traversing a demand graph of buyers. Instead, ERM starts with low prices for all resources and iteratively increases these prices until no buyer is willing to pay a higher price.
- Different to state of the art solutions, ERM is further not only designed to compute equilibrium prices but also to compute *differential prices* without violating the $(1+\varepsilon)$ market equilibrium. Previous approaches can only include fixed transaction costs (e.g., taxes or fixed differential) for buyers and resources [36]. They additionally treat

identical copies of multiple instances as independent resources and are thus unable to capture price differentiation. ERM is the first approach to exploit SPLC utilities, which allow the price differential to vary depending on utility, budget and resource types.

- We evaluate ERM with uniform, normal and Pareto distributions to represent a good match of heavy and non-heavy tailed distributions to represent various buyer utility behaviors (Section 5.4.1). Our results show that ERM, in fact, offers orders of magnitude more ($10\times$ to $100\times$ in most cases) revenue for the market provider than state-of-the-art approaches. At the same time, it is proven to be fair by achieving the Nash Social Welfare (NSW), and this property increases as the problem complexity grows. Further, ERM captures dynamic prices based on supply and demand of resources. Finally, while convex program solvers do not scale beyond 50 buyers, we show that ERM scales well with increasing numbers of buyers and resource types, and can compute a solution within seconds, even for large scenarios with 500 buyers and resource types.

## 5.2 ERM Problem

Our market features multiple buyers interested in one or multiple resource types. Further, the market provider may offer multiple instances of each resource[2]. Here, a buyer's utility for different instances of the same resource type is not necessarily the same since an instance can run in different environments with different impacts on its performance. Buyers express their interest in *utility* (definition 2.2) for the resources. Section 2.2 presents the essential concepts of the market.

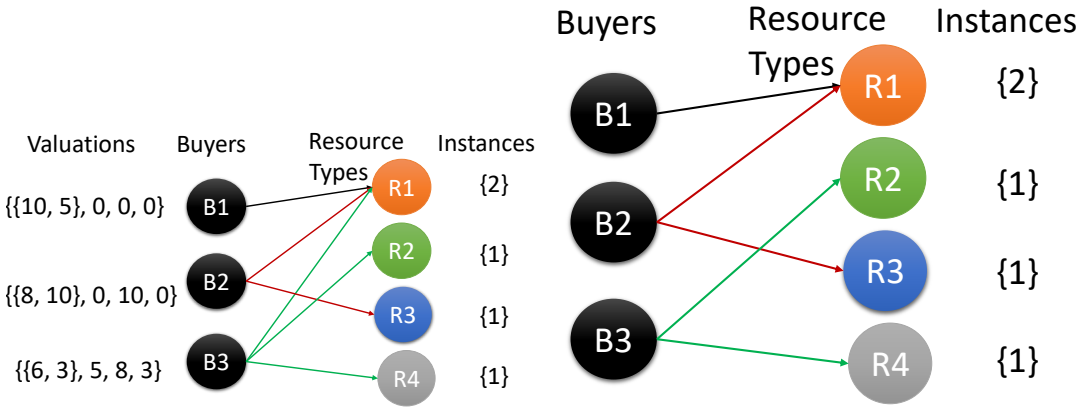
The goal of the market is to allocate resources to successful buyers. The critical challenges are:

- The market $\mathscr{M}$ should have the *market clearing* property [21].
- The resources offered in $\mathscr{M}$ are indivisible.
- The market should offer rational equilibrium prices [33]. For a rational input, rational equilibrium implies the existence of a rational optimal solution with bitsize bounded polynomially by the input size [119]. For such a rational optimal solution, we can employ linear programming solvers instead of slow convex solvers, which adds to the practicality of ERM [119].
- The market should offer *differential prices*.

Figure 5.1 depicts a simple example of four resource types (R1, R2, R3, R4) with R1

[2]The market provider could play the role of a middleman who procures resources from resource providers and pays them their due for a fee [84].

having two instances and the rest having just one instance. Figure 5.1a shows three buyers arrive with an initial valuation of the three resource types. We can observe in Figure 5.1b that an ideal allocation technique would assign R2 and R4 to buyer B3 since she is the only one expressing interest in it, albeit at a very low price. Moreover, all the three buyers are interested in the two instances of R1 and R1 is subsequently sold to the two highest bids. Fig. 5.1b illustrates that all the resources have been sold at differing prices and that all the buyers have received at least one of the resources they bid for. Next, we formalize the problem.




(a) Buyer's resource valuations (b) Allocation

Figure 5.1: Example depiction of market clearing allocation.

Let $B = \{1,2,\ldots,n\}$ be a set of $n$ buyers. Let $M = \{1,2,\ldots,m\}$ be the set of resource types, where $j$ represents the $j^{th}$ resource type. Also, $|B| = n$ and $|M| = m$. For each resource type $j$, there are $\ell_j$ instances. Let $u_{ijk}$ be the utility derived by the buyer $i$ for buying the $k^{th}$ instance of resource type $j$. Let $x_{ijk}$ be the amount of $k^{th}$ instance of resource type $j$ allocated to the buyer $i$. Let $e_i$ be the total endowment or money of the buyer $i$.

The utility of a resource can be modeled with a variety of different approaches. Cloud resource utilities are usually modeled by *Constant Elasticity of Substitution* (CES) utility functions. The Leontief utility is a well known CES function family but has an irrational equilibrium [115, 120]. In real systems, an irrational equilibrium cannot be written in polynomial bits [33]. Linear utility functions (convex combinations of resources) on the other hand are simple and widely used. However, they fail to model the concept of decreasing marginal utility (i.e., diminishing returns). SPLC utility functions address these limitations.

In our context, consider a simple service with two components, namely a web server and a database. The total utility of the service is the additive of the PLCs of both components. The market equilibrium of SPLC is rational [33]. Hence, in our market, we assume that

buyers utility is SPLC.

The goal of social choice theory is to perform a fair division of resources. There are three approaches—utilitarian, egalitarian and Nash Social Welfare (NSW). A utilitarian approach favors buyers with large utility. Conversely, an egalitarian approach favor buyers with small utility, while NSW has a Pareto outcome. The latter implies that the allocation can only be improved at the expense of a buyer.

The goal of this work is to perform resource allocation such that the NSW of all buyers, i.e., the geometric mean of their utilities, is maximized. Taking logarithm, we obtain the following convex program [30]:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \Big(\prod_{i=1}^{n} u_i^{e_i}\Big)^{\frac{1}{n}} \\ \text{subject to} \quad & \sum_{i=1}^{n}\sum_{k=1}^{\ell_j} x_{ijk} \le \ell_j, \qquad \forall j \in M, \\ & \sum_{j=1}^{m}\sum_{k=1}^{\ell_j} x_{ijk}(p_j + q_{ijk}) = e_i, \qquad \forall i \in B, \\ & x_{ijk} \in \{0,1\}, \quad \forall i \in N, j \in M, k \in \ell_j. \end{aligned} \tag{5.2.1}$$

Let $\boldsymbol{p}$ be the $m \times 1$ vector of prices and $\boldsymbol{x}$ be $n \times m \times \ell_j$, the vector of allocations, where the entry at $(i, j, k)$ is $x_{ijk}$, a boolean decision variable indicating whether buyer $i$ is allocated the $k^{th}$ instance of resource type $j$. Let $\mathscr{M} = (\boldsymbol{p}, \boldsymbol{x})$ be the market and $\boldsymbol{p}$ is market clearing prices and allocation at equilibrium $\boldsymbol{x}$. In other words, let $p_j$ be the *base price* associated with the resource type $j$ and, similarly, let $q_{ijk}$ be the *price differential*, i.e., the extra money paid by the buyer $i$ for the $k^{th}$ copy of the $j^{th}$ resource type. The goal of this work to find both the base price and the differential price for each resource type such that the allocation satisfies Eq. (6.2.1).

The total utility $u_i$ is given by $u_i = \sum_{j=1}^{m}\sum_{k=1}^{\ell_j} u_{ijk} \cdot x_{ijk}$, where $u_{ijk}$ is the utility of buyer $i$ for the $k^{th}$ instance of resource type $j$. The objective function is to maximize the NSW of all buyers. The first constraint ensures that the allocations never exceed $\ell_j$ instances of resource type $j$. The second constraint guarantees that the total price paid by the buyer never exceeds his total money. The third constraint implies only integral allocation.

We assume the following, consistent with [84]:

- The supply of resource units is limited i.e., $\ell_j \neq \infty, \forall j \in M$.

- There is at least one buyer for all the resource instances, i.e., $u_{ijk} > 0, \forall j \in M, \ell_j \in j$. The prices are determined such that all instances are sold as long as there are enough buyers, i.e., if required, prices are fixed in such a way that even buyers with less money are satisfied if there is no other buyer with more money available for a resource.

In this chapter, we design $\mathscr{M}$ as $(1+\varepsilon)$-approximate equilibrium if the budget is satisfied within a factor $(1+\varepsilon)$ i.e., $\sum_{j=1}^{m}\sum_{k=1}^{\ell_j} x_{ijk}(p_j+q_{ijk}) = (1+\varepsilon)e_i, \forall i \in B$ so that $\boldsymbol{p}$ and $\boldsymbol{x}$ are *market clearing*. The notations are summarized in Table 5.1.

| Symbol | Description |
|---|---|
| $B$ | Set of buyers |
| $M$ | Set of resources |
| $u_{ijk}$ | Utility of buyer $i$ for $k^{th}$ instance of $j^{th}$ resource |
| $u_i$ | Total utility of buyer $i$ |
| $e_i$ | Total budget or endowment buyer $i$ |
| $\ell_j$ | Total number of resource $j$ instances |
| $x_{ijk}$ | Boolean variable denoting allocation of $k^{th}$ instance of $j^{th}$ resource to buyer $i$ |
| $q_{ijk}$ | Differential price $k^{th}$ instance of $j^{th}$ resource to buyer $i$ |
| $p_j$ | Price of resource $j$ |
| $\boldsymbol{p}$ | Price vector of all resources |
| $\boldsymbol{x}$ | Allocation vector of all buyers |

Table 5.1: A summary of notations used for proposing ERM.

## 5.3 ERM Algorithm

To simplify the computation, we take the logarithm of the objective function of Eq. (5.2.1), and subsequently, have

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{i=1}^{n} e_i \cdot \log u_i \\ \text{subject to} \quad & \sum_{i=1}^{n}\sum_{k=1}^{\ell_j} x_{ijk} \leq \ell_j, \qquad \forall j \in M, \\ & \sum_{j=1}^{m}\sum_{k=1}^{\ell_j} x_{ijk}(p_j + q_{ijk}) = e_i, \qquad \forall i \in B, \\ & x_{ijk} \in \{0,1\}, \quad \forall i \in N, j \in M, k \in \ell_j. \end{aligned} \tag{5.3.1}$$

Eq. (5.3.1) is popularly known as an *Eisenberg-Gale* or Fisher market [26, 30, 34]. In the above formulation, $p_j$ is the *base price* of the $j^{th}$ resource type, while $q_{ijk}$ is the *differential price* for the $k^{th}$ instance of resource $j$, i.e., additional money paid by a buyer $i$ to achieve higher utility. $e_i$ is the initial endowment, i.e., the maximum amount of money that the buyer $i$ is willing to spend.

We define *bang-per-buck* $\alpha_{ijk}$ of the $k-th$ copy of $j$ as the ratio of the utility of a resource instance to its price, i.e., $\alpha_{ijk} = \dfrac{u_{ijk}}{p_j}$. $\alpha$ describes how much a buyer benefits from buying the resource instance at this price. Further, the *rate* for each buyer, $r_i = \dfrac{u_i}{e_i}$, is the average utility a buyer $i$ expects to gain unit money, expressed as the ratio of total utility to the money of $i$ [84]. Equilibrium exists only if Eq (5.3.1) has an optimal solution which implies that the *Karush-Kuhn-Tucker* (KKT) conditions are satisfied [21, 34]. Section 2.1.2 presents the fundamentals of KKT. The KKT conditions for Eq (5.3.1) are:

- $\forall j \in M, p_j \geq 0$ and $\forall i \in N, j \in M, k \in \ell_j, q_{ijk} \geq 0$ which implies that both the price and the differential price are non-negative.
- $\forall j, p_j \geq 0 \implies \sum_{i=1}^{n}\sum_{k=1}^{\ell_j} x_{ijk} = \ell_j$ which implies complete allocation of all instances of the different resource types are sold.
- $q_{ijk} > 0 \implies x_{ijk} = 1$ which implies that price discrimination will affect only the successful allocations.
- $\forall i \in B, j \in M, k \in \ell_j, p_j + q_{ijk} \geq \dfrac{e_i \cdot u_{ijk}}{u_i}$ and if $x_{ijk} > 0 \implies p_j + q_{ijk} = \dfrac{e_i \cdot u_{ijk}}{u_i}$ and implies that at optimality, the actual bang-per-buck of a buyer $i$ is at most $u_i$.

The pseudo-code of ERM is presented in Algorithm 5.1.

**Algorithm 5.1** ERM auction

**Require:** Utility vector $u_i, \forall i \in B$

1: initialize()
2: **while** $\forall i \in N, s_i > 0$ or $\forall j \in M, \sum_{i=1}^{n} \sum_{k=1}^{\ell_j} x_{ijk} = \ell_j$ **do**
3: buildDemandGraph()
4: transferWalk()
5: $\forall j \in M, p_j \leftarrow (1+\varepsilon) \cdot p_j$
6: **if** $\forall i \in B, j \in M, k \in \ell_j, \exists y_{ijk} = 1$ **then**
7: $x_{ijk} \leftarrow 1$
**return** $\forall i \in B, j \in M, k \in \ell_j, x_{ijk} \forall j \in M, p_j$

**Algorithm 5.2** initialization

1: **procedure** INITIALIZATION($u$)
2: $\forall j \in M, p_j \leftarrow 0$
3: $\forall i \in B, j \in M, k \in \ell_j, q_{ijk} \leftarrow 0$
4: $u_i \leftarrow \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{\ell_j} u_{ijk}$
5: **for** $j \leftarrow 1, m$ **do**
6: $p_j = \min u_{ijk}, \forall i \in N, k \in \ell_j$
7: $b, w \leftarrow \arg\min u_{ijk}, \forall i \in N, k \in \ell_j$
8: $y_{bjw} \leftarrow 1$
**return** $\boldsymbol{p}, \boldsymbol{y}$

The basic idea of ERM (Algorithm 5.1) is to allocate the most demanded resource instances of each buyer in an iterative way, thereby reducing buyer surplus (i.e., increasing the money spent for each buyer). First, ERM performs initialization to ensure that there is at least one buyer for each resource instance, i.e., the prices are *market clearing* (Algorithm 5.2). Here, for every resource instance, we find the buyer with the *minimum* utility for the resource instance, and this utility is considered as the *base price* of the instance (line 6). The corresponding buyer is temporarily allocated with the instance (line 7). We call this allocation the *temporary allocation*. In every round $t$, we increase the base price of each resource instance $k$ by the factor $(1+\varepsilon)$, i.e., $p_{j,k,t-1} = \dfrac{p_{j,k}}{(1+\varepsilon)}$.

In general, the *surplus* of a buyer $i$ is calculated as

$$s_i = e_i - \left( \sum_{j=1}^{m} \sum_{k=1}^{\ell_j} (p_j + q_{ijk}) \cdot x_{ijk} + \sum_{j=1}^{m} \sum_{k=1}^{\ell_j} \left( \frac{p_j}{(1+\varepsilon)} \right) \cdot y_{ijk} \right) \tag{5.3.2}$$

In other words, the surplus is the remaining money $i$ has left after spending on resource allocations. After initialization, the surplus of each buyer is reduced by the sum of his valuations for his temporarily allocated resource instances.

Then, ERM performs resource allocation iteratively over multiple rounds, where each round can be divided into three stages. First, we determine for every buyer with non-zero surplus her most desired resource instance, i.e., the resource instance with a maximum bang-per-buck. With these instances as input, ERM second generates the *demand graph* (Algorithm 5.3), where each buyer is represented by a node, and initially no edges exist. We then construct edges in $G$ as follows (line 6): each resource instance $k$ of type $j$ was previously temporarily allocated to a buyer, say $i$. In the demand graph, we now create an edge e($i'$,$i$) between each buyer $i'$ for whom the $k^{th}$ instance of $j$ is the most-wanted resource instance, and $i$. In other words, $i'$ is interested in the resource allocated to $i$.

**Algorithm 5.3** buildDemandGraph

```
1: procedure BUILDDEMANDGRAPH(B)
2:     ∀i ∈ B, v_i ∈ V
3:     E ← ∅
4:     for i ← 1, n do
5:         if ∃i' and i' ≠ i, y_{i'dc} then
6:             edg ← (i, i')
7:             E ← E ∪ {edg}
       return G = (V, E)
```

During this allocation, particularly for high-demand resources, two or more buyers might be interested in the same resource instance. To resolve this issue, ERM as a third stage traverses the demand graph in a DFS style from each buyer node. On each edge it traverses, it transfers the surplus between the two adjacent buyers $i'$ and $i$, if $i'$ is willing to pay more (i.e., the *differential price*) for the $k^{th}$ instance of $j$ than $i$. This is the case, if the bang-per-buck $\alpha_{i',j,k}$ is higher than $r_i$, i.e., the required price (increased by the price differential) is still attractive for $i'$ (line 3 in Algorithm 5.6), and the $i$ has a sufficient amount of surplus (line 6). If both requirements are satisfied, the allocation of $k$ will be changed to $i'$, and the price of the resource is updated appropriately. The money previously invested by $i$ into $k$ is added to the surplus of $i$, while the surplus of $i'$ is reduced by the updated price. After traversing the edge—regardless of whether or not the allocation of $k^{th}$ instance of $j$ was changed or not—the edge is removed from the graph. This process is repeated until the DFS traversal reaches a leaf node. Note that

1. by removing edges during traversal, ERM also avoids loops in the graph, which is essential to ensure the feasibility of ERM in distributed settings [36].

2. a resource can be allocated to different buyers throughout the round due to the itera-

tive DFS processing of buyer nodes.

3. after traversing all buyer nodes, there will be no edges left in the graph, and all resources will be allocated, i.e., they are either allocated to a new buyer or remain with the old one (in the extreme case, with the temporarily allocated buyer during initialization).

**Algorithm 5.4** transferWalk

```
1: procedure TRANSFERWALK(G = (V,E))
2:     for i ← 1,n do
3:         β_i ← DFS(i)
4:         srcEdg ← i
5:         for ∀η ∈ β_i do
6:             dstEdg ← η
7:             transferSurplus(srcEdg,dstEdg,d,c)
8:             srcEdg ← η
       return G = (V,E)
```

ERM then repeats these three stages iteratively for the next most-wanted resource instances. In each iteration, it again increases the resource base price by the factor $1+\varepsilon$. Here, the rationale is that the market provider has two significant goals. One is to sell all resource instances (market clearing prices), which is achieved by the guarantee of all resource instances being allocated as long as at least one buyer is interested in them at an arbitrarily low price due to ERM's initialization procedure. The second one is to increase revenue. This is achieved by increasing the prices of resources in every round. Here, informally, ERM transfers resources from an already allocated buyer to another buyer with a higher surplus for a higher price. This iterative process is performed until all buyers have zero surplus or all resource instances are allocated, at which time ERM converges.

**Theorem 5.1 (ERM approximation ratio)** ERM converges with $(1+\varepsilon)$ market equilibrium.

**Proof** First, we prove that ERM maintains the constraints of Eq(5.3.1). The initialization (Algorithm 5.2) performs allocation of each resource instance of all resource types with a minimum price $\frac{p_j}{(1+\varepsilon)}$ (line = 7). Hence, before starting the auction, all instances are already allocated (i.e., sold). Since we initialize with minimum prices, the sum of all the resource prices is less than the total budget. In this case, all the constraints are satisfied.

In an auction round, the allocation is modified when there is a buyer $i'$ with enough surplus $s_{i'}$ to buy the $k^{th}$ resource instance of $j$ (line 9, Algorithm 5.6). Also, the negative surplus of buyers is one of the terminating conditions of ERM. This implies that the budget constraint is met in every round. The differential price is calculated based on the KKT

**Algorithm 5.5** transferSurplus

[htpb]

1: **procedure** TRANSFERSURPLUS($G = (V,E)$)
2: Calculate *srcSurplus* and *dstSurplus* for $i'$ and $i$ using Eq (5.3.2)
3: **if** $\alpha_{idc} > r_i$ **then**
4: $q_{idc} \leftarrow e_i \cdot \frac{u_{idc}}{u_i}$
5: **if** $dstSurplus > (p_d + q_{idc})$ **then**
6: adjustAllocation $(i,d,c)$
7: $edg \leftarrow (i',i)$
8: $E \leftarrow E \setminus \{edg\}$

**Algorithm 5.6** adjustAllocation

1: **procedure** ADJUSTALLOCATION($i,d,c$)
2: **if** $\alpha_{idc} > s_i$ **then**
3: $q_{idc} \leftarrow e_i \cdot \frac{u_{idc}}{u_i}$
4: **else**
5: $q_{idc} \leftarrow 0$
6: $curPrice \leftarrow p_d + q_{idc}$
7: **if** $s_i > curPrice$ **then**
8: $prevPrice \leftarrow p'_j + q_{i'jk}$
9: **if** $y_{i'dc}$ or $curPrice > prevPrice$ **then**
10: $x_{idc} \leftarrow 1$
11: $x_{i'dc} \leftarrow 0$
12: $y_{i'dc} \leftarrow 0$

conditions. Moreover, for every buyer $i$, the sum of base price and differential price always satisfies $(p_j + q_{ijk}) = \frac{u_{ijk}}{u_i}$.

Finally, if there are unallocated resource instances due to low demand, then the initial temporary allocation is made permanent(line 6 of Algorithm 5.1). Hence, ERM maintains the conditions of $(1+\varepsilon)$ throughout the running time.

Let $u_{min} = \min u_{ijk}$ and $\theta = \sum_j^m \ell_j$. Since the prices are updated from $u_{min}$ to $\sum_i e_i$, there are at most $R = 1 + \frac{\theta}{u_{min}} \log \frac{\sum_i e_i}{u_{min}}$ rounds. The initialization and final allocation can be done in $n\theta$ operations. The depth-first search takes $\mathcal{O}(n)$ operations and total surplus transfer

can be achieved in $\mathcal{O}(n)$ operations as well. The demand set of the buyers can be found in $\mathcal{O}(n \log \theta)$ if we use a max-heap and hence the demand graph construction takes at most $\mathcal{O}(n^2 \log \theta)$. The total time complexity of ERM is, therefore, $\mathcal{O}((n^2 \log \theta + n)R)$.

## 5.4 ERM Evaluation

### 5.4.1 Methodology

Generally, the internal pricing strategies of cloud service providers are private and, to the best of our knowledge, there exists no open real-world data for evaluating market mechanisms. Hence, most existing works perform custom simulations and generate the data based on probability distributions [24, 35, 81, 86]. Codenotti *et al.* [114] evaluate different market mechanisms by generating a desirability matrix and using Constant Elasticity Substitution (CES) utility functions to generate utilities for the buyers. Similarly, we use an SPLC function to determine buyer utilities in combination with the desirability matrix.

Initially, we generate a *desirability matrix* which represents the buyers' resource instance valuations. We generate the matrix elements based on different properties such that the sum of each row is 1. Subsequently, we generate the utility matrix for the complete market as a product of the desirability matrix and uniform values in the interval $[0, 100]$. The pseudocode in Algorithm 5.7 presents our approach.

**Algorithm 5.7** Data set generator

**Require:** $N$ buyers, $M$ resource, *max_instances* and *min_instances*

1: **for** $i \leftarrow 1, n$ **do**
2: **for** $j \leftarrow 1, m$ **do**
3: $\gamma \leftarrow random(min\_instances, max\_instances)$
4: $\forall i \in N, j \in M, k \in \gamma$ Generate $d_{ijk}$ based on generator type such that $\sum_{k}^{\gamma} d_{ijk} = 1$ and $\sum_{j}^{|M|} d_{ij} = 1$
5: $\forall i \in N, j \in M, k \in \gamma, u_{ijk} \leftarrow d_{ijk} \cdot random[0, 100]$
**return** Utility matrix $u_{ijk} \in UM$

We obtain different desirability matrices as follows:

- Uniform generator: The matrix elements are uniformly distributed in the interval $[0,1]$.
- Subset generator: In this approach, initially we generate the subset $J$ of $M$ such that $|J| \leftarrow random[1,|M|]$. The desire of buyers outside of $J$ is 0. For example, if we have 5 resources and $|J| = 3$, then the buyers are interested only in items in $J$ and for other items the utility is 0.

In our evaluation, we use the combination of these two generators so that there is at least one buyer for the resource, i.e., $\exists i'$ such that $u_{i'jk} \neq 0$, which is consistent with our market assumptions. The sum of the coefficients of the two generators is 1. To determine the utilities, we consider the following probability distributions:

- Uniform distribution: This is a simple and widely used probability distribution for evaluating market equilibrium [115].
- Normal distribution: As the number of samples $n \to \infty$, non-heavy-tailed distributions converge to normal [70]. Hence, evaluation on a normal distribution guarantees similar behavior as in other non-heavy-tailed distributions.
- Pareto distribution: This is a well known heavy-tailed distribution frequently used to model income and wealth [121].

We consider the following scenarios:

- Fixed resource types and varying buyers: In this scenario, the number of resource types is fixed throughout the experiments. In our case, we fix the number of resource types to 500 and increase the number of buyers.
- Fixed buyers and varying resource types: In this case, we fix the number of buyers to 100 while resource types are increased.

For each scenario, we generate multiple instances for each resource type uniformly in the range of $[1, 10 \mod m]$, where $m$ is the number of resource types. For instance, if we have 500 resource types, then each resource type will have instances in the interval $[1,5]$. Hence, the problem size and complexity increase with the number of resource types. Our evaluation goals are:

- To determine the revenue increase `ERM` yields for the market provider. To compute the increase, we compare the revenue obtained by `ERM` to that obtained by a Vickrey auction. The Vickrey auction or second-price auction [100] is a well-known auction algorithm used in trading. In this auction a resource is allocated to the highest bidder; however, the winner only pays the second highest bid to encourage truthful bidding.
- To evaluate the effect on the NSW of tuning *epsilon*, i.e., the price increase in each round of our auction algorithm.

- To evaluate the effect of varying *epsilon* on the running time of ERM. Note that for auctioning time-critical services, our solution ideally needs to compute within a few seconds.

## 5.4.2 Results

### 5.4.2.1 Provider Revenue

Figures 5.2 and 5.3 show the revenue factor (ratio of profit obtained by ERM to profit obtained by the Vickrey auction) for different distributions, different values of $\varepsilon$ and different scenarios.

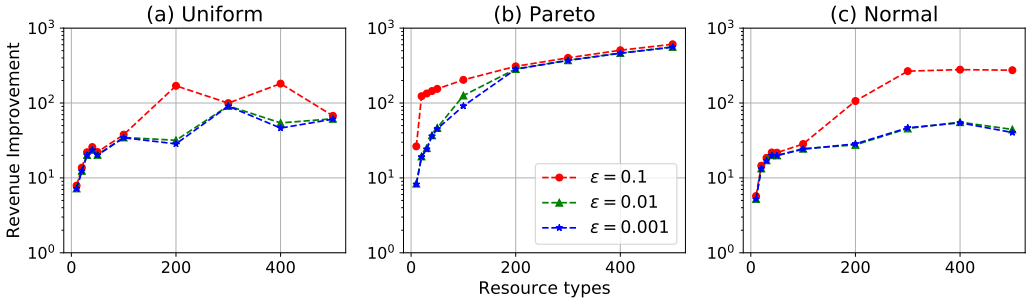



Figure 5.2: Fixed Buyers: Normalized revenue improvement factor for different distributions and different values of $\varepsilon$.

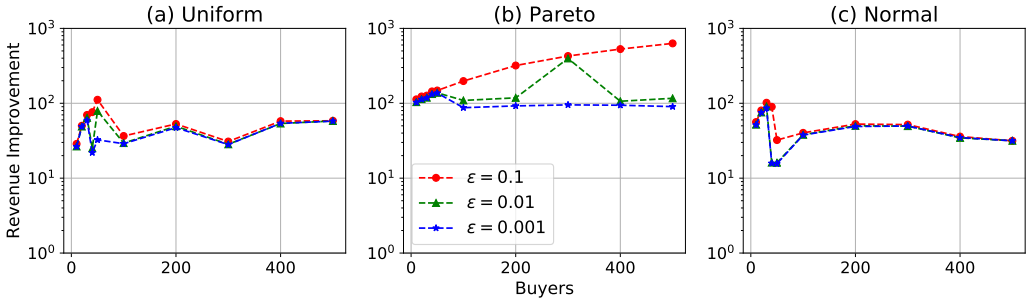



Figure 5.3: Fixed Goods: Normalized revenue improvement factor for different distributions and different values of $\varepsilon$.

We observe that the revenue of ERM is at least one order of magnitude higher than for Vickrey auctions. Most improvement factors for normal and uniform distributions are in

the range of 10× to 100×, with several occasion where ERM achieves more than that. Further, ERM scales better than the Vickrey auction in this regard, as the ratio increases with an increase in resources. Additionally, we observe that the revenue improvement of ERM is the highest for the Pareto distribution, caused by its heavy-tailed nature. Here, the utilities of the highest and second-highest bidder may significantly differ. In a Vickrey auction, the allocated buyer pays the second highest price, whereas ERM will calculate a price closer to the valuation of the highest bidder. Finally, as a larger $\varepsilon$ also implies additional budget allowance among buyers, increasing $\varepsilon$ offers more revenue for varying resource types. Figure 5.4 summarizes the improvement factors for each distribution in a CDF.

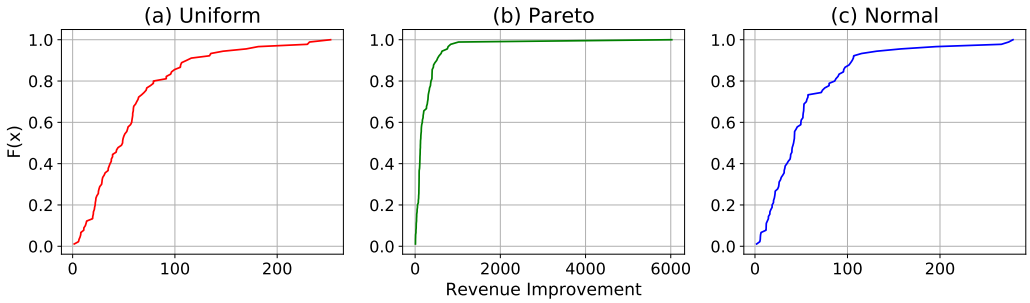



Figure 5.4: Revenue improvement CDF for different distributions

#### 5.4.2.2 Social Welfare

Simultaneously Nash Social Welfare (NSW) increases as we scale up buyers and resources in the market as shown in Figures 5.5 and 5.6, again supporting the scalability of our approach.

We further observe that a smaller $\varepsilon$ results in a *slightly* better fairness, caused by smaller increases of the price in each round of the auction and thereby the possibility to fulfill more buyer demands—instead of overpricing resources for more buyers with higher $\varepsilon$ values. Note, however that (i) the difference between the $\varepsilon$ values is rather small (except for the Pareto distribution), and (ii) that there is practically no difference between $\varepsilon = 0.01$ and $\varepsilon = 0.001$.

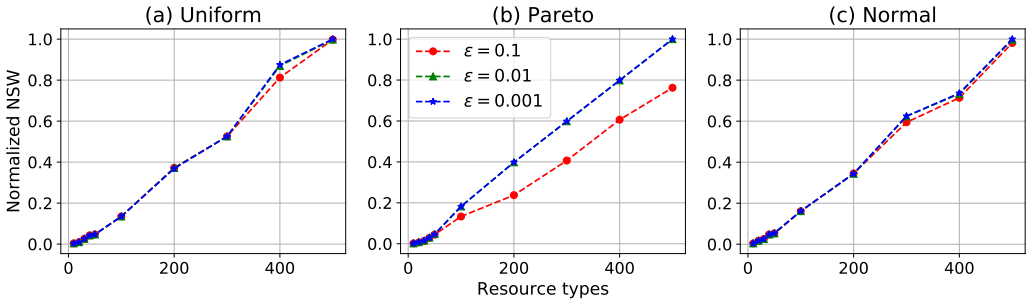



Figure 5.5: Fixed Buyers: Normalized Nash social welfare (NSW) for different distributions and different values of $\varepsilon$.

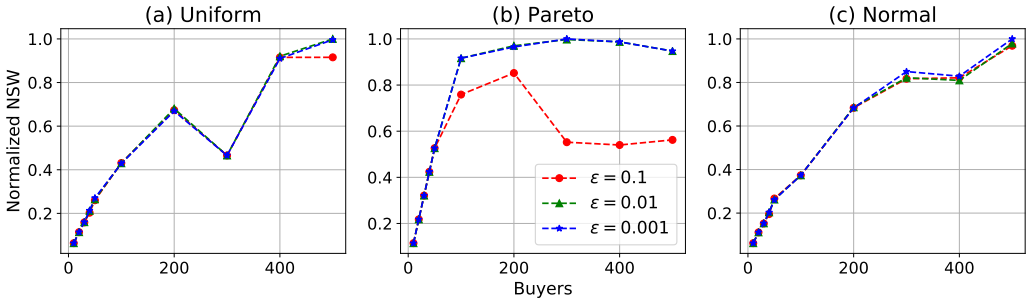



Figure 5.6: Fixed Goods: Normalized Nash social welfare (NSW) for different distributions and different values of $\varepsilon$.

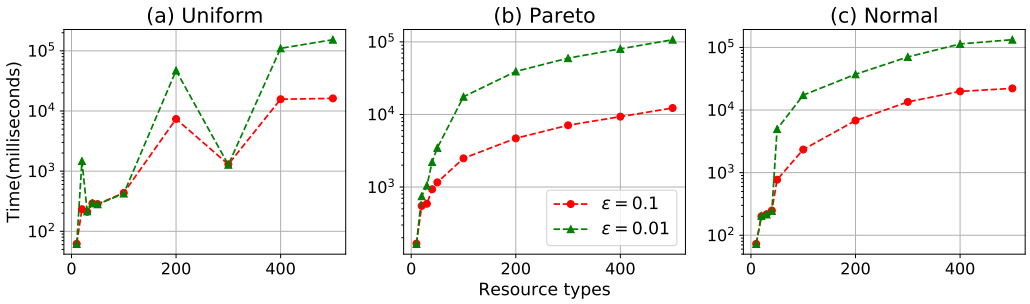



Figure 5.7: Fixed Buyers: Running time of ERM for different distributions and different values of $\varepsilon$.

#### 5.4.2.3 Performance of ERM

As one use case of ERM is edge resource auctioning, our algorithm needs to converge fast. Therefore, in Figures 5.7 and 5.8 we evaluate the computation time of ERM, again under the influence of $\varepsilon$. Since prices are increased by the factor $(1+\varepsilon)$ in each round, $\varepsilon$ co-determines the number of rounds ERM requires to converge. Thus, smaller $\varepsilon$ values will result in slower convergence as ERM needs to compute additional rounds of auctions.

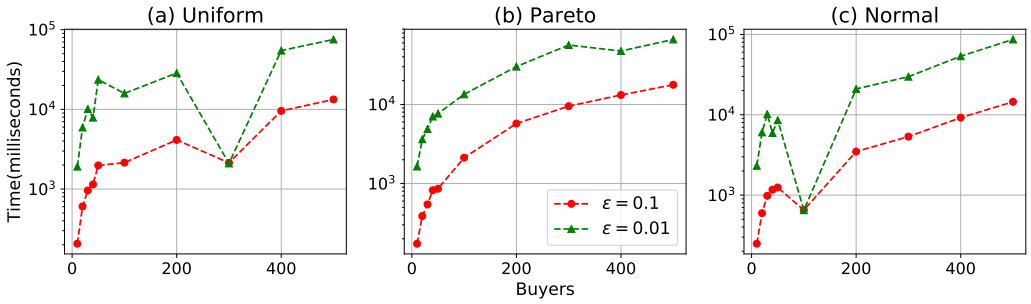



Figure 5.8: Fixed Goods: Running time of ERM for different distributions and different values of $\varepsilon$.

Our key observation is that for time-critical auctions, $\varepsilon$ should be set to larger values. Here, $\varepsilon = 0.1$ yields computation times of less than one second for smaller markets (up to 40 buyers and 100 resource types), and less than 10 seconds for larger markets (up to 500 buyers and resource types). When changing $\varepsilon$ to 0.01, ERM will take over one minute to compute a solution for the larger markets. This might not be sufficient for time-critical service provisioning.

## 5.5 Summary

In this chapter, we proposed ERM, a market for edge resource allocation. Different from state-of-the-art solutions, ERM ensures that resource prices are *market clearing* based on supply and demand; maximizes NSW fairness among buyers with $(1+\varepsilon)$ market equilibrium; and is based on auction performs integer allocation to reflect a realistic case of allocating indivisible resources of the market with SPLC utilities.

Another significant feature of ERM is that it facilitates price discrimination by computing *differential prices* dynamically contrary to state-of-the-art solutions, enabling market providers to provide differential services. The experiments demonstrate substantial revenue

improvement compared to Vickrey auctions. While standard convex solvers do not scale beyond 50 buyers, our experiments show that ERM scales with the number of resource types and buyers and with an appropriate value of $\varepsilon$ can be applied to edge computing with less loss of NSW.

# Chapter 6

# ARPP Online Pricing

Pricing algorithms proposed in previous chapters 4 and 5 are *offline* algorithms—algorithms with complete knowledge of the input. This chapter presents OFM, an *online* Fisher market where the input is revealed after the prices for the current time interval are computed.

Initially, we motivate the need for *online* NSW maximization in *cloud* and *edge*. Subsequently, we formalize an *online* Fisher market for varying buyers and resources and introduce our OFM adversarial model. Eventually, we propose the OFM algorithm. Finally, we evaluate and present experimental results.

## Contents

## 6.1 Introduction

Currently, most of the services and applications offered by service providers are virtualized and hosted in data centers. Due to many benefits over hardware appliances—such as its inherent scaling flexibility—virtualization is a primary enabler for new paradigms such as edge computing and network function virtualization (NFV). Edge computing is gaining in popularity for a variety of applications. For instance, edge computing can support the trend to virtualize network functions [104] by a) providing these virtualized services close to the user for alleviating the latency concerns [4, 105, 106]; b) avoid the need to redirect traffic to cloud services [107];and c) support the scale up/down of services to meet user demand [108]. IoT applications such as those on vehicles, smart-homes, and factories could benefit from leveraging edge computing [109] to avail virtualized services such as pre-processing the large amount of data collected.

*Cloud and edge service or resource pricing*[3] is one of the important challenges for any cloud and edge service providers. Low resource prices affect profit while higher prices drive away customers. Hence, pricing affects profit and customer experience simultaneously. As a consequence, resource pricing has captured the attention of both academia and industry [63]. Currently, cloud service providers employ a "pay-as-you-go" or *fixed* pricing model. In this model, prices are computed based on usage time. For instance, Amazon EC2 instances are billed hourly. Fixed pricing maximizes the revenue only if every customer behavior is *well-defined* and arrivals are *temporally invariant* [12]. For example, consumers spending more money on weekends than weekdays is an example of a well defined behavior. The temporal invariance implies constant customer arrivals. However, these conditions are not valid in cloud computing because both customer demands and arrivals are ad-hoc [13]. Moreover, the physical capacity of cloud resources is finite [14]. Most often, the default fixed pricing favors cloud service providers contractually [19]. Further, resource prices are uniform regardless of demand and scarcity [113]. Hence, cloud service providers cannot benefit from an additional profit during high demand periods and suffer a loss during low demand periods. Hence, fixed pricing neither maximize the revenue of cloud service provider nor fair towards customers.

The cloud computing community is exploring alternative pricing schemes such as *posted pricing* and *dynamic pricing* [18–20] to address the earlier mentioned fixed pricing issues. In posted pricing, customers appear sequentially, and the seller publishes prices. Customers procure resources only if prices are acceptable. Hence, this approach is alternatively known as *leave-it-or-take-it* pricing. However, posted pricing approaches does not guarantee *equilibrium* or *market-clearing* prices—prices guarantee complete allocation of all resources [21].

[3] In this work we use the terms services and resources interchangeably

In dynamic pricing, resource prices reflect the supply and demand of the resource, i.e., the higher the demand, the higher the prices. Most often dynamic pricing leads to efficient resource utilization and satisfies user demands [13]. There are some efforts from the industry to move towards dynamic pricing. For instance, Amazon introduced the concept of *spot pricing*. In spot pricing, a user can specify maximum prices he is willing to pay, and instances are allocated until spot instance prices are within the maximum price. However, spot instance prices are not determined based on market demand [20].

Generally, cloud service providers are profit driven. Hence, prices maximize revenue for the service provider. Alternatively, maximizing social welfare especially in cloud computing improves not only overall system efficiency but also assures a better user experience [13]. Further, maximizing social welfare is appropriate for both public and private clouds [22]. Therefore, maximizing social welfare is beneficial for both cloud service providers and users.

There are three types of social welfare namely *utilitarian*, *egalitarian* or *max-min fairness* and *Nash social welfare*(NSW). In utilitarian, the aim is to maximize overall utility of customers and service providers. Most of the works [20] in the Cloud regarding dynamic pricing are *utilitarian* and favor customers with higher utility. Conversely, the *egalitarian* goal is to maximize the minimum utility of the customers and rewards customers with lower utility. In literature [25], NSW is the *Pareto outcome* between utilitarian and egalitarian approaches. Further, NSW is *scale-free*, i.e., optimal allocation is independent of the scale of each customer's utility. Hence, maximizing NSW is appropriate for a cloud and edge environment. Further, cloud customer demands are *online* and hence require *online* pricing [13]. However, designing *online* market maximizing NSW is non-trivial due to the following challenges:

(C1) Customer valuations are not independent and identically distributed [42] and vary dynamically, and decisions are often time-critical [13].

(C2) Cloud resources are usually virtualized and therefore rendered intangible [18]. These cloud resources need to be allocated as a whole, i.e., a fractional allocation (such as one half of a virtual machine (VM)) is impractical in cloud environments.

We believe that there exists no solution that solves these challenges. Therefore, in this work, we introduce `OFM`, an extended online Fisher market for dynamically pricing cloud resources. `OFM` yields the following contributions:

(i) `OFM` maximizes NSW of providers and customers while achieving market clearing prices. In particular, it achieves a balance between efficiency and fairness. While state-of-the-art solutions typically follow a utilitarian approach [82, 83, 86], NSW has been proven to be fair to all customers and the providers [25].

(ii) OFM allows for the online allocation of resources. State-of-the-art Fisher markets typically employ *offline* algorithms, i.e., algorithms with complete knowledge about the input [26, 27, 84, 122, 123] that includes the buyer utilities as well as the number of buyers and resources. In the online version of a Fisher market, the resources freshly appear in each round and allocation decisions as well as price calculations are performed simultaneously in an irrevocable fashion [47, 48, 88]. Both are not aligned with the cloud reality in which customers dynamically arrive at the market and procure a fixed set of available resources. OFM fits the needs of a cloud resource market. In contrast to existing solutions, OFM can converge and allocate resources without aprior knowledge about the number of buyers and their utilities in each allocation round.

(iii) OFM ensures the integer allocation of resources. State-of-the-art solutions typically round fractional allocations to the nearest integer solution. Unfortunately, the resulting rounding difference, also called as *integrality gap* is unbounded for a market, i.e., the difference grows with the number of buyers [30]. The OFM formulation implies only integer allocation since constraints force buyers to pay either the full price of the resource or nothing (Theorem 6.1).

(iv) Online algorithms usually are analyzed with worst-case input. However, observed data in real systems is not worst-case input at all times. Devanur and Hayes [124] proposed a random permutation model for this scenario in an online algorithm. Also, this model is more generic compared to independent and identically distributed data (iid) with both known and unknown distributions [125]. Moreover, algorithms for random permutated models provide better performance for an arbitrary input [125]. Furthermore, this enables us to capture changing customers and resources offered over a period. The adversarial model of OFM is not only stricter than state-of-art solution [49] but also more optimistic than [48].

(v) Online convex optimization (OCO) [126] methods are widely used for online problems with *Lipschitz continuous* objective functions. The NSW objective of Fisher market neither guarantee integer allocation nor *Lipschitz continuous* [127] . Hence, OFM uses an equivalent convex program proposed in [122, 123]. This formulation not only maximizes NSW but also guarantee integer allocation (Theorem 6.1). Even this formulation is also not *Lipschitz continuous*. OFM apply a simple relationship between concave and convex functions and convert the objective to a convex function by flipping the sign of the objective function. Informally, OFM performs optimization in the opposite direction of the objective. The prices computed by both concave and convex functions are the same. The equilibrium prices form unit simplex [122]. Hence, the value of the prices is in $[0, 1]$ all the time.

(vi) OFM uses online mirror descent algorithm [126] with unnormalized negative entropy as the regularization function. In the first stage, OFM predicts the current price based on the previous period price. The input is revealed after the prediction and OFM will

perform an update on the regularization function and projects back to the current objective function. This projection minimizes the Bregman divergence of the regularization function. The OFM updates are closed-form expressions and are computationally efficient. Hence, OFM can scale for a large number of buyers and customers. Intuitively, OFM picks the resource prices near the previous optimal resource allocation. We measure the performance of OFM using *regret*. Regret is the difference between an online player and a static player with hindsight information.

Generally, the internal pricing mechanisms are not made public due to the fear of losing competitive edge over other service providers. We believe that there is no openly available cloud pricing data of service providers. Balserio *et al.* [128] generate a stochastic dataset based on Google AdX [4] real data to evaluate their AdX placement algorithms. Further, Bateni *et al.* [49] [5] extend this dataset for their evaluation.

Each dataset consists of varying advertisers and impression types. The number of advertisers and impressions are different in each dataset. Also, the utilities of the advertisers are sparse in one data set while dense in another dataset. We evaluate OFM in two scenarios namely fixed resources and varying resources. In fixed resources, we extend the dataset by distributing buyer utilities normally and uniformly. The experimental results clearly show the convergence of OFM with a prediction accuracy of about 95% of optimal prices on datasets namely AdX, normal and uniformly distributed data.

For evaluating varying resource scenario, we generate the number of resources offered at each time instance by randomly sampling CPU demand without replacement from the Google cluster data trace [129]. We employ prediction schemes to predict the resources offered. We use time series models, mean of the previous resource offered and resource offered in the last interval. Our evaluation shows the superiority of ARIMA (Autoregressive Integrated and Moving Average) model over other regarding prediction accuracy. However, merely using the information of last offered resources can reduce computational time without affecting overall performance.

Furthermore, we perform experiments to determine the impact of buyers on OFM. The results clearly show that OFM computation time is in the order of *microseconds* for a large number of buyers. Hence, OFM is a candidate for the edge computing market where there is a need for quick and irrevocable price computations.

[4] https://developers.google.com/ad-exchange/

[5] https://bitbucket.org/florinciocan/fairresourceallocation.git

## 6.2 OFM Problem

Let $\mathcal{N} = \{1,2,\ldots,n\}$ be a set of $n$ buyers indexed by $i$ i.e., $i$ represents the $i^{th}$ buyer. Let $\mathcal{R} = \{1,2,\ldots,m\}$ be the set of resources indexed by $j$, i.e., $j$ represents the $j^{th}$ resource. Let $x_{ij}$ be the fraction of allocation of the $j^{th}$ resource to the $i^{th}$ buyer. Let $u_{ij}$ be the utility derived by $i^{th}$ buyer for $j^{th}$ resource and $u_i = \sum_{j=1}^{m} u_{ij} \cdot x_{ij}$ be the total utility derived by the $i^{th}$ buyer. Also, let $b_i$ be the budget of the $i^{th}$ buyer i.e., , total endowment or money of buyer $i$.

In a cloud, maximizing social welfare not only improves overall system efficiency but also assures a better user experience [13]. In social choice theory, there exist three types of social welfare—*utilitarian*, *egalitarian* and *Nash Social Welfare* (NSW). In a *utilitarian* approach, the goal is to maximize total utility of buyers and customers while an *egalitarian* approach maximizes the minimum utility of the customers. As a result, buyers with a higher utility are favored in the former approach while the later favors buyers with smaller utility. NSW achieves a *Pareto* outcome between utilitarian and egalitarian. Conversely, NSW achieves a balance between efficiency and fairness [25]. Moreover, NSW is *scale-free*—optimal allocation is independent of the scale of each customer's utility. As a consequence, there is no incentive to inflate or deflate utility.

We assume the following, consistent with [84]:

- The supply of resources is limited.
- There is at least one buyer for all the resources, i.e., $u_{ij} > 0, \forall j \in \mathcal{G}$. The prices are determined such that all resources are sold as long as there are enough buyers. In other words, prices are computed in such a way that even buyers with less money are satisfied if there is no other buyer with more money available for a resource.

Theoretically, NSW is the geometric mean of the utilities of all the buyers, i.e., $(u_1^{b_1} \cdot u_2^{b_2} \cdots u_n^{b_n})^{\frac{1}{n}}$. Additionally, as discussed before, in the context of cloud computing the notion of fractional allocation does not have practical application. Therefore, existing solutions typically round the fractional part to the next integer value [130] as the difference between actual and rounded values are usually negligible. This difference is known as the *integrality gap*. However, Cole *et al.* [30] show that for a Fisher market, this rounding approach leads to suboptimal performance due to an unbounded integrality gap (integrality gap increases with the number of buyers). Therefore, OFM considers resources to be indivisible which implies that allocation of a resource is either 0 or 1. If we take the logarithm, then the maximization of NSW reduces to an Eisenberg-Gale or Fisher market [26]. The convex

program is as follows:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{i=1}^{n} b_i \log u_i \\ \text{subject to} \quad & u_i = \sum_{j=1}^{m} u_{ij} \cdot x_{ij}, \quad \forall i \in \mathscr{N}, \\ & \sum_{i=1}^{n} x_{ij} \leq 1, \qquad j \in \mathscr{R}, \\ & x_{ij} \in \{0,1\}, \qquad \forall i \in \mathscr{N}, j \in \mathscr{R}. \end{aligned} \tag{6.2.1}$$

Unfortunately, Eq. (6.2.1) cannot be used directly in an online setting to address challenge C2 due to the following reasons:

- Most of the online convex optimization algorithms assume that the objective function is *Lipschitz* continuous. Our objective function, however, is not only *non-Lipschitz* continuous (the rate of change of the function is not constant) but also non-convex.
- Even if we circumvent *non-Lipschitz* continuity by shifting the valuations to a range $\{1, \cdots, K+1\}$, where a number $K > 0$ then we will end up as a linear factor in regret bound leading to low performance (around less 20%) [131].
- Integer allocation is not guaranteed as discussed above.

Cole *et al.* [123] provide a convex program equivalent to 6.2.1 as follows:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \left( \frac{\prod_i \prod_j u_{ij}^{x_{ij}}}{\prod_j p_j^{p_j}} \right) \\ \text{subject to} \quad & \sum_{i=1}^{n} y_{ij} = p_j, \qquad \forall j, \\ & \sum_{j=1}^{m} y_{ij} = b_i, \qquad \forall i, \\ & x_{ij} \in \{0, p_j\}, \quad \forall i, j, p_j \leq 1. \end{aligned} \tag{6.2.2}$$

Here, $p_j$ is the price associated with resource type $j$. Also, $y_{ij}$ is the amount paid by the buyer $i$ for the $j^{th}$ resource. The first constraint implies that the total amount paid for a resource by all buyers never exceeds resource price. The second constraint guarantees that the total amount paid by the buyer is within his overall budget $b_i$.

**Lemma 6.1 (Equivalence of OFM objective function)** Eq. (6.2.1) and Eq. (6.2.2) maximize the NSW. Further, the logarithm of the objective function Eq. (6.2.2) maximizes NSW.

**Proof** The equivalence of Eq. (6.2.1) and Eq. (6.2.2) is proven in [123]. The logarithm of the objective $\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ij}\log u_{ij} - p_j \log p_j)$. This is the objective function of the convex program proposed by [122] for determining equilibrium prices. The proof of NSW maximization for this objective function can be found in [122, 132]. Hence, the objective $\sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij}\log u_{ij} - p_j \log p_j)$ is NSW maximizing.

If we take the logarithm of the objective of Eq. ( 6.2.2) and apply an additional constraint to enforce integer allocation [123], we get the following convex program:

$$\begin{aligned}
\underset{x}{\text{maximize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij}\log u_{ij} - p_j \log p_j) \\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = p_j, \qquad \forall j \in \mathscr{R}, \\
& \sum_{j=1}^{m} x_{ij} = b_i, \qquad \forall i \in \mathscr{N}, \\
& x_{ij} \in \{0, p_j\}, \quad \forall i \in \mathscr{N}, j \in \mathscr{R}, p_j \leq 1.
\end{aligned} \tag{6.2.3}$$

The third constraint implies that the buyer will either pay the full price or nothing.

**Theorem 6.1 (OFM objective function property)** The convex program (6.2.3) maximizes NSW and performs integer allocation.

**Proof** The objective of Eq. (6.2.3) is the logarithm of Eq. (6.2.2). Hence, by using lemma 6.1, Eq. (6.2.3) maximizes NSW. The constraint $x_{ij} \in \{0, p_j\}$ and $\sum_{i=1}^{n} x_{ij} = p_j$ implies that there can be only one buyer among others with $x_{ij} = p_j$, while for the rest of buyers $i' \neq i$, $x_{i'j} = 0$. Substituting, we get, $\sum_{j=1}^{m} x_{ij}\log u_{ij} - x_{ij}\log x_{ij}$ and the following

convex program:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij}\log u_{ij} - p_j \log p_j) \\ \text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = b_i, \quad \forall i \in \mathscr{N}. \end{aligned}$$

The objective function of convex program 6.2.3 is not only *non-Lipschitz continuous* but also concave. Hence, online convex optimization based methods cannot be applied directly. Therefore, we perform the following steps before applying online convex optimization methods:

- We apply a simple relationship between concave and convex functions and convert the objective to a convex function by flipping the sign of the objective function. Then the goal is changed to minimize the convex function (i.e., $\max g(x)$ and $\min -g(x)$ are equivalent). Informally, we perform optimization in the opposite direction of the objective.
- In a Fisher market, equilibrium prices form a unit simplex, i.e., the sum of normalized prices of all goods is equal to 1 [122]. Furthermore, the unit simplex not only reduces computational complexity but also implicitly enforces the constraint of Eq. (6.2.4). Still, the optimality of the solution is not affected. Hence, we address the *non-Lipschitz continuous* nature of the program by restricting the input set to a unit simplex.

Finally, the convex program for `OFM` is as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m}(x_{ij}\log u_{ij} - p_j \log p_j) \\ \text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = b_i, \quad \forall i \in \mathscr{N}, \\ & x_{ij} \geq 0, \quad \forall i \in \mathscr{N}, j \in \mathscr{R} \end{aligned} \tag{6.2.4}$$

The above formulation is the minimization of the *Kullback-Leiber (KL) divergence* [133] between price and utility [123]. In summary, Eq. (6.2.4) addresses the challenge C2 described in Section 6.1. In the subsequent section, we develop the adversarial model for addressing the challenge C1. The following Table 6.1 summarizes the notations used in this section.

| Symbol | Description |
|---|---|
| $N$ | Set of buyers |
| $R$ | Set of resources |
| $u_{ij}$ | Utility of buyer $i$ for $j^{th}$ resource |
| $u_i$ | Total utility of buyer $i$ |
| $b_i$ | Total budget or endowment buyer $i$ |
| $x_{ij}$ | Boolean variable denoting allocation of $j^{th}$ resource to buyer $i$ |
| $p_j$ | Price of resource $j$ |

Table 6.1: A summary of notations used in OFM.

## 6.3 OFM Adversarial Model

Online algorithms, unlike their offline counterparts, do not have complete information about the input. Hence, for analyzing the performance of online algorithms, adversarial models are proposed in the literature. The basic idea of an adversarial model is to present input which negatively affects the performance of the algorithm. For instance, quicksort performs poorly when the input is already sorted. Most of the proposed online algorithms are pessimistically designed towards a *fully adversarial* model where the input data is provided for the worst-case.

In real scenarios, often input is *well-behavedness*. For instance, in sponsored search auctions, the query keywords are classified into *head* and *tail* keywords based on the frequency of their appearance. Head keywords frequently appear while tail keywords appear rarely. The tail keywords are the reason for the competitive edge of search engines over traditional media such as television advertisements. However, tail keywords are harder to optimize [124]. Furthermore, both pessimistic and stochastic adversarial models cannot model tail behavior.

Devanur and Hayes [124] proposed a random permutation model for modeling tail keyword behavior and account for the behavior of observed data in an online algorithm. We can observe a similar tail behavior in cloud computing as well. For instance, the Amazon AWS EC2 service is more popular than the AWS Elastic search service [134]. Hence, a random permutation model is more suitable for cloud computing due to its ability to model tail behavior and its generic nature. In essence, the random permutation model is *sampling without replacement*, while independent and identically distributed data with known and unknown distributions are *sampling with replacement* [124]. In sampling with replacement, samples drawn are independent of each other while they are dependent in sampling

without replacement. Hence, the random permutation model is more generic compared to independent and identically distributed. Importantly, any algorithm that works on a random permutation model will work for independent and identically distributed models [135] and provide better performance for arbitrary input [125]. Finally, this enables us to model customers and resource dynamics over a period. Hence, we use the random permutation adversarial model for the online fisher market proposed in [125].

Briefly, `OFM` random permutation model not only captures tail behavior but is also more generic compared to stochastic models. Simultaneously, it is less pessimistic compared to *fully adversarial*. In summary, `OFM` adversarial model captures the *well-behaveness* of the data compared to the state-of-the-art adversarial models for the online Fisher market [48, 49].

Formally, we define our random permutation model as follows [125]: The adversary picks an input consisting of $n = |\mathcal{N}|$, $m = |\mathcal{R}|$, $m_i, \forall i \in \mathcal{N}$ and $d_{ij} \forall i \in \mathcal{N}, j \in \mathcal{R}$. A permutation $\pi$ of $\mathcal{G}$ is chosen uniformly at random. In round $t$, the buyer's utility inputs are $u_{ij}^t = d_{i\pi(j)}$ and the budget of the buyer $i$ is $b_i = e_{\pi(i)}$.

Hazan *et al.* [136] define *regret* as the difference between the total cost of the current decision and the best single decision with the benefit of hindsight. Informally, regret is the performance measure between an online player and a static player with hindsight information. Let $\ell_t$ be the instance of Eq. (6.2.4) at period $t$ or in other words the value of the *loss function* at period $t$. Formally, we denote the cumulative regret of the objective function as follows [137]:

$$\boldsymbol{R^o} = \sum_{t=1}^{T} \ell_t(x_t) - \min_x \ell_t(x_t) \tag{6.3.1}$$

In summary, the design goals of `OFM` are:

(D1) Find prices $p_j, \forall j \in \mathcal{R}$ at every time instance $t$ for varying buyers and resources which maximize *Nash social welfare* and approximately market clearing.

(D2) *Minimize* the regret $\boldsymbol{R^o}$.

The design goal D1 addresses challenge C2 while design goal D2 address the challenge C1 as described in Section 6.1. In the subsequent section, we present the `OFM` algorithm to achieve above design goals. In the next section, we present the `OFM` algorithm to achieve the above goals.

## 6.4 OFM Algorithm

The design goal D2 can be achieved by choosing an approximate online convex optimization algorithm. At every time instance, the numbers of buyers and resources are dynamic—a typical scenario in the cloud [13]. Further, the computation of equilibrium prices for an offline algorithm using convex optimization solvers is not scalable [114]. Therefore, several works [27, 28] compute approximate equilibrium market prices. Hence, even for an offline algorithm, computing market equilibrium prices is non-trivial and especially true in the online scenario with a one-time decision and varying demand and supply as well. Thus, achieving D1 for OFM is non-trivial and very challenging.

At every time instant $t$ OFM will be in either of following scenarios:

- Fixed resource set: The number of resources offered by the provider(s) is constant over a period with varying buyers.
- Varying resource set: The number of resources offered by the provider(s) and buyers both varies at every time instance.

With a fixed resource set, since the offered resources are constant, the prices depend solely on the buyer utility at that time instant $t$. If the objective function can be solved using a closed expression, then the resulting algorithm not only computationally efficient but also scales with the numbers of buyers. We can obtain a closed-form expression for the OFM objective in this scenario for finding market equilibrium prices.

**Lemma 6.2 (OFM closed form expression)** The closed form expression for computing prices of the OFM objective function with $m$ fixed resources is given by $x_{ij} = \dfrac{b_i u_{ij}}{\sum_{j=1}^{m} u_{ij}}$.

**Proof** Let $\alpha_i$ be the dual variable associated with each constraint in equation (6.2.4). Now we derive the closed form expression for $x_{ij}$ which minimizes the value of equation (6.2.4)

$$\mathscr{L}(x,\alpha) = \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}\log x_{ij} - x_{ij}\log u_{ij} + \sum_{i=1}^{n}\alpha_i\big(b_i - \sum_{j=1}^{m} x_{ij}\big)$$

Let $g(\alpha)$ be the dual function and by definition, we have

$$\begin{aligned} g(\alpha) &= \sup_{x\in\Delta} \mathscr{L}(x,\alpha) \\ \nabla_x \mathscr{L}(x,\alpha) &= \log x_{ij} + 1 - \log u_{ij} - \alpha_i \end{aligned}$$

Let $\mu_{ij} = \log u_{ij} + \alpha_i$, then

$$= \log x_{ij} + 1 - \mu_{ij}$$

At optimality, $\nabla_x \mathscr{L}(x,\alpha) = 0$

$$\implies x_{ij} = \exp\{\mu_{ij} - 1\}$$

Substituting $x_{ij}$ in $g(\alpha)$, we have

$$g(\alpha) = \sup \sum_{i=1}^{n} b_i \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{m} e^{\mu_{ij}-1}$$

Eliminating $\mu_{ij}$,

$$= \max \sum_{i=1}^{n} b_i \alpha_i + \sum_{i=1}^{n}\sum_{j=1}^{m} u_{ij} e^{\alpha_i - 1}$$

$g$ is maximized when $\nabla g = 0$, therefore,

$$b_i = \sum_{j=1}^{m} u_{ij} \exp(\alpha_i - 1)$$

Finally, the maximum value of $g(\alpha)$ is

$$\max g(\alpha) = \sum_{i=1}^{n} (\log b_i - \log \sum_{j=1}^{m} u_{ij})$$

According to strong duality theorem, at optimality the primal and dual objective have same value. If we substitute $x_{ij} = \dfrac{b_i u_{ij}}{\sum_{j=1}^{m} u_{ij}}$, equation (6.2.4) achieve minimum value. The dual of (6.2.4) is:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{i=1}^{n} b_i \alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{m} u_{ij} e^{\mu_{ij}-1} \\ \text{subject to} \quad & \mu_{ij} - \log u_{ij} \le \alpha_i, \quad \forall i \in \mathscr{N}, j \in \mathscr{R}, \\ & \alpha_i \ge 0, \quad \forall i \in \mathscr{N}. \end{aligned} \tag{6.4.1}$$

At optimality,i.e., $x_{ij} = \dfrac{b_i u_{ij}}{\sum_{j=1}^{m} u_{ij}}$ then $\alpha_i = 1$. The value of $x_{ij}$ is same for the maximization of Eq(6.2.3) and readers can see at [123].

Hence, for fixed resource set, we can compute the prices efficiently and quickly.

The varying resource set is a generic problem compared to the fixed resource scenario and hence, more challenging. Furthermore, a typical real case scenario in cloud subletting.

In cloud subletting, a user can monetize her unused or under-utilized resources by subletting to other users [138]. The service provider can act as a broker on behalf of a user for subletting resources. At every period, interested users can submit under-utilized resources to the service provider. The service provider can sublet for a specified period. Hence, the service provider lacks complete knowledge of the total number of resources offered at every instance. However, it is necessary to predict the number of resources offered approximately. For instance, in a sponsored search auction, if the number of queries is unknown apriori, then the competitive ratio of the online algorithm will be bounded away from 1 even for a small number of bidders and query keywords [124]. Hence, Devanur and Hayes [124] advocate knowing the number of queries offered approximately to improve the performance of the online algorithm.

In OFM, the number of resources offered is revealed only after the current prices are computed. However, OFM has access to the past data, and there will be patterns and trends in supply and demand such as periodical changes from low to high demand and vice versa. Time series analysis is widely used to predict future data based on such trends [139]. OFM updates the time series model based on the input data and performs a prediction of the number of resources offered in the next time instance. Here, OFM can employ different prediction models such as ARIMA, but also more straightforward approaches such as simple moving averages and immediate past values.

Once OFM has predicted the number of resources, then the scenario is similar to a fixed resource set and OFM finds the new prices near to the previous instance optimal prices. In this way, OFM achieves its second goal.

We introduce the notation and definitions used by the OFM algorithm. The objective function of OFM is not only convex but also time variant with varying buyers and resources. The equilibrium prices form a unit simplex [122]. Let $\Delta_t$ be the unit simplex constructed from the set $\mathscr{X}$ at time $t$. Formally, $\Delta_t = \left\{ x \middle| \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} = 1 \right\}$. The cardinality of $\Delta_t$ changes only when there is a change in the number of resources offered. That is, $|\Delta_t| \neq |\Delta_{t'}|$ if $m_t \neq m_{t'}$. Let $\Delta$ be the set of all unit simplex till period $T$. Formally, $\Delta = \{\Delta_t | t \leq T\}$.

Let $\boldsymbol{x}^t$ be the price matrix (column vector of size $m \times n$) which is the price buyers pay for the resources at round $t$, i.e., $\boldsymbol{x}^t = (x_{ij}^t, \forall x_{ij} \in \Delta_t)$. Let $m_t$ be the number of resources offered at time $t$ and assume that $k$ past values are available, i.e., $m_{t-k}, m_{t-k-1}, \ldots, m_t$.

Online mirror descent (OMD) is one of the widely used algorithms for online convex optimization [126, 140, 141]. The basic idea of OMD is to perform an update on the dual space of the regularizing function and to project the update on the convex decision set using appropriate distance generating functions iteratively. The regularization function not only

improves the stability but also lowers the regret bounds [126]. Furthermore, OMD has following characteristics [141]:

- Nearly optimal regret can be achieved even for a full adversary (worst case input data) by using OMD for any convex online learning problem by defining an appropriate distance generating function. Hence, can be applied to any convex learning problem and therefore, OMD is considered as *universal* [141].
- The OMD update is a first order method (involves only the slope of the function). Hence, most of the times the updates are simple and computationally efficient.

Hence, `OFM` employs OMD. To apply OMD, we require a distance generating function, which is dependent on the geometry of the objective function. The *Bregman divergence* [142] is one of the widely used distance generating functions. It is used to measure the distance between the function and the first order Taylor expansion of the function (tangent). Formally, let $h : \Omega \times ri(\Omega)$ be a continuously differentiable convex function and let $\boldsymbol{p}$ and $\boldsymbol{q}$ be the two points on $h$ with gradient $\nabla \boldsymbol{p}$ and $\nabla \boldsymbol{q}$ respectively. Then, the Bregman divergence is defined as follows:

$$B_h(\boldsymbol{p}, \boldsymbol{q}) = \phi(\boldsymbol{p}) - \phi(\boldsymbol{q}) - \langle \nabla \boldsymbol{q}, \boldsymbol{p} - \boldsymbol{q} \rangle \tag{6.4.2}$$

For a Fisher market, an unnormalized negative entropy function is used as the regularization function [132, 143]. Hence, `OFM` uses unnormalized negative entropy as the regularization function, i.e., $h(x) = x \log x - x$ and the corresponding Bregman divergence expression can be found in Appendix A.2. The pseudo-code of `OFM` is presented in Algorithm 6.1. `OFM` works in two stages. The prices are predicted in the first stage and the objective function is updated in the later stage. Let $m_0$ be the resources offered at time $t = 0$. Initially, `OFM` determines the number of resources randomly and the corresponding unit simplex. Furthermore, prices for all resources are initialized to $\frac{1}{m_0}$ such that the unit simplex property $\sum_{j=1}^{m_0} p_j = 1$ is satisfied.

For every period $t$, `OFM` first predicts the number of resources (line 5). If the number of resources offered is not the same as in the previous period $t-1$, then either new resources are added or existing resources have been removed. The removal of a resource is straightforward and involves only updating the length of the set $\Delta$ (line 12). Moreover, it does not violate budgets. Let $\delta$ be the difference between the number of resources predicted during $t$ and the actual number of resources offered in $t-1$, i.e., $\delta = m'_t - m_{t-1}$. Let $\sigma_{t-1}$ be the sum of all prices at instance $t-1$. `OFM` performs the following during the addition of new resources ($\delta > 0$).

- Compute the new prices for $m_t$ using OMD update. These resources are old resources.

- Compute the difference between the sum of current prices of old resources and $\sigma_{t-1}$. The new prices are initialized with the value $\frac{\sum_{j=1}^{m_{t-1}} x_j - \sigma_{t-1}}{\delta \cdot t}$ since equilibrium prices form unit simplex. If new resources are introduced later, then they should be initialized with low prices to satisfy the unit simplex property. Otherwise, will lead to the budget violation. Therefore, it is necessary to penalize the offering of new resources at a later period.

OFM computes the difference and updates the length of the current price vector $x_t$(line 6). The prices are predicted before the input is revealed (line 13) using algorithm 6.2. Once prices are predicted, the function $\ell_t$ along with the input parameters $m, n$, and $u_{ij}$ are revealed to OFM. These parameters are used to update the online mirror descent of the OFM algorithm. Further, we compute the optimal prices for the current period using the closed-form expression provided in lemma 6.2(line 15).

**Algorithm 6.1** OFM algorithm

**Require:** $\ell_t, m_t, n_t$

1: $m_0 \leftarrow random()$
2: $\forall j \in m_0, x_0 = \frac{1}{m_0}$
3: $\Delta = \Delta_{m_0}$
4: **for** $t \leftarrow 1, T$ **do**
5: $\quad m'_t \leftarrow predictResources(predict, t)$
6: $\quad$ **if** $m'_t \neq m_{t-1}$ **then**
7: $\quad\quad \delta \leftarrow m'_t - m_{t-1}$
8: $\quad\quad$ **if** $\delta > 0$ **then**
9: $\quad\quad\quad \forall k \in \delta, x_t[k] \leftarrow \frac{\sum_{j=1}^{m_{t-1}} x_j - \sigma_{t-1}}{\delta \cdot t}$
10: $\quad\quad\quad$ Set $|x_t| \leftarrow \max\{m_{t-1}, m'_t\}$
11: $\quad$ **else**
12: $\quad\quad$ Set $|x_t| \leftarrow \min\{m_{t-1}, m'_t\}$
13: $\quad x_t \leftarrow OFM - MD(predict, t)$
14: $\quad$ Observe $\ell_t$
15: $\quad x_t \leftarrow \arg \ell_t(x^*)$
16: $\quad$ **if** $m_{t-1} \neq m_t$ **then** $\Delta \leftarrow \Delta \cup \{\Delta_{m_t}\}$
17: $\quad$ onlinePrice($update, \ell_t, x_t$)
18: $\quad$ predictResources($update, m_t$)

Algorithm 6.2 is the pseudo-code for online price prediction. The primary function of Algorithm 6.2 is to predict the current prices based on the previous period $t$. OMD performs

the price prediction in two stages. They are:

- In the first stage, an update is performed on the regularization function $h(x)$. Let $y_t$ be the update for $h$ at instance $t$. In OFM, the update rule is given by $\nabla h(y_{t+1}) = \nabla h(x_t) - \eta \nabla_t$, where $\nabla_t$ is the gradient of $\ell_t$.
- Once $y_{t+1}$ is calculated, then the prediction at time $t+1$, i.e., $x_{t+1}$ is the projection on $\Delta$ that minimizes Bregman divergence between points in $\Delta$ and $y_{t+1}$ on the function $h$. Formally, we can write as the following optimization problem:

$$\underset{x \in \Delta}{\text{maximize}} \quad B_h(x, y_{t+1}) \tag{6.4.3}$$

  Eq. (6.4.3) is minimized if $\nabla B_h(x, y_{t+1}) = 0$. This implies that $x_{t+1} = y_{t+1}$. The proof can be found in Appendix A.2.

In OFM, the structure of the objective function $f$ is time-invariant but the number of consumers, resources offered and utility are time variant. Hence, we get a closed-form expression for updates and predict the prices for the next instance instantly as soon as the input is revealed in Algorithm 6.2(line= 5).

**Algorithm 6.2** onlinePrice algorithm

```
1: procedure ONLINEPRICE(state, f', x')
2:     x_0 ← 1/e
3:     for t ← 1, T do
4:         if state == predict then return x_t
5:         if state == update then
6:             ℓ_t ← ℓ'
7:             x_t ← x'
8:             ∇_t = ∇ℓ_t(x_t)
9:             x_{t+1} = e^{(log x_t − η∇_t)}
```

In summary, OFM performs the computation of market equilibrium prices for the resources and prediction of resources offered at every period. The computation of optimal prices requires $m \cdot n$ steps. Furthermore, the minimum prices need $n$ steps. Computing the slope using previous prices and current price prediction require $n$ steps each. Hence, the time complexity for every round is $\mathcal{O}(m \cdot n)$. The total time complexity for $T$ periods is given by $\mathcal{O}(T \cdot (\boldsymbol{m} \cdot \boldsymbol{n}))$ where $\boldsymbol{m} = \max m_t, t < T$ and $\boldsymbol{n} = \max n_t, t < T$.

**Lemma 6.3** (OFM **regret bound)** The regret bound of OFM is $\boldsymbol{R^o} \leq 2\sqrt{2T \log \boldsymbol{n}}$.

**Proof** In OFM, $\Delta$ is a simplex and the sum of optimal offline prices (equilibrium prices) never exceeds 1 [122]. Hence, the slope of OFM is bounded even though it is logarithmic. Let $\|\nabla_t\|$

be the dual norm of the slope at period $t$ and for $\Delta$, $\|\nabla_t\| \leq 1$. Let $\boldsymbol{n} = \max n_t, t < T$ be the maximum number of resources offered in OFM. By substituting these values in [126, §5.4], we get the regret bound $\boldsymbol{R^o} \leq 2\sqrt{2T \log \boldsymbol{n}}$.

## 6.5 OFM Evaluation

### 6.5.1 Methodology

Resource pricing is of strategic importance for any service provider. Generally, the internal pricing mechanisms are private due to the fear of losing a competitive edge over other service providers. We believe that cloud pricing data of service providers are unavailable. Balserio *et al.* [128] generate stochastic dataset based on Google AdX real data to evaluate their AdX placement algorithms. Bateni *et al.* [49] extend this dataset by augmenting volatile information to model the sensitivity of shocks due to social and news trends such as negative publicity about the resources in the news.

Each dataset consists of varying advertisers (6 to 101) and impression types (7 to 406). The number of advertisers and impressions are different in each dataset. Also, the utilities of the advertisers are sparse in one data set while dense in another dataset. Arrivals are assumed to be the Ornstein-Uhlenbeck process. An Ornstein-Uhlenbeck process is a diffusion process for modeling the velocity of a particle in Brownian motion and widely used in mathematical finance to model market prices and volatility. The parameters for Ornstein-Uhlenbeck process are estimated on the dataset presented in [128] and emulated arrival data is generated by preserving statistical properties of the real dataset. The mean values of impression type $j$ and advertiser $i$ is considered as utility $u_{ij}$. The estimation methodology can be found in [49].

We divide our evaluation into two scenarios namely the case of a *fixed set of resources* and the case of a *varying set of resources*. The latter scenario is more generic and challenging than the earlier. We measure the regret and competitive ratio of OFM. Regret is the distance between our OFM online algorithm objective without hindsight and an optimal algorithm with hindsight. Conversely, the competitive ratio is the ratio of OFM online solution and optimal offline solution.

#### 6.5.1.1 Fixed resource set

It is evident from Section 6.2 that buyer utility affects the resource prices but not the number of buyers. The modified AdX dataset [49] cannot be directly applied for evaluating OFM

since the probability distribution is limited to an Ornstein-Uhlenbeck process. We perform the following steps to modify the dataset for evaluating OFM.

- Each impression type is treated as a resource and advertisers are treated as buyers. Let $\Lambda_j^t$ be the Ornstein-Uhlenbeck arrival rate of the resource $j$ at time interval $t$. In our case, we treat mean values of impression type $j$ and advertiser $i$ as a base utility $u_{ij}^*$ and generate a new utility $u_{ij}$ for a resource and a buyer at every period as the product of base utility and Ornstein-Uhlenbeck arrival rate of the resource, i.e., $u_{ij}^t = u_{ij}^* \Lambda_j^t$. In this way, we ensure that the volatility of buyer utilities in every period $t$. In OFM random permutation model, the expectation of the data varies at each time interval. Hence, maintaining volatility captures the random permutation scenario in the evaluation.
- In a real marketplace, buyers arrive with different budgets, and it is essential to incorporate them in our OFM evaluation. In the dataset of [49], the budget of an impression type is calculated based on $C_j$, the total number of the impression type of $j$ which advertisers are willing to buy. We also calculate budget along similar lines. In our case, the budget of buyer $i$ is given by $b_i = \frac{\sum_j u_{ij}}{\left|\{j : u_{ij} > 0\}\right|}$.
- The goal of OFM is to handle utilities from a different distribution. Hence, we use different distributions for generating utilities, namely uniform and normal distributions. A uniform distribution is a simple and widely used distribution and the utility is generated in the interval $[0,1]$ uniformly. According to the central limit theorem, non-heavy tailed distribution over a period will converge to a normal distribution [70]. Hence, evaluation on a normal distribution guarantees similar behavior as in other non-heavy-tailed distributions.

In summary, the AdX dataset is modified to evaluate with different probability distributions of a buyer's utility for a fixed set of resources scenario.

#### 6.5.1.2 Varying resource set

Consider the scenario where the resources vary every time instance. Hence, the OFM solution set is frequently modified. Since the number of resources offered at the current time instance is revealed only after current prices are predicted, this scenario is not trivial. Furthermore, this is a typical real case scenario in cloud subletting. In cloud subletting, the users can monetize their unused or underutilized resources by subletting to other users [138]. The service provider can act as a broker for subletting resources. At every period, interested users can submit underutilized resources to the service provider. The service provider can sublet for a specified period. Hence, the service provider lacks complete knowledge of the total number of resources offered at every instance.

The AdX dataset cannot be used for this scenario since both buyers and resources are fixed. Hence, we perform a trace-based simulation for OFM evaluation in this scenario.

We use a Google cluster data trace [129] of around $12.5k$ machines collected over a period of 29 days in a Google data center. In this dataset, jobs arrive and VMs are allocated for the execution. Each job has different CPU requirement and hence, different VMs are allocated which eventually leads to different CPU usage and CPU demand. In other words, the CPU demand is not uniform for all time intervals and depends on the demands of incoming jobs. We use this demand information to simulate the demand behavior of resources in OFM. We generate the number of resources offered at each time instance by randomly sampling CPU usage without replacement from the $41GB$ dataset since random permutation model is sampling without replacement. Furthermore, we assume that service providers introduce more resources during high demand time periods. We perform time series analysis and use Box Jenkin's method to build an ARIMA model to predict the CPU usage for future prediction. OFM uses this prediction information to set the prices for current time instance.

The results of OFM evaluation for both the scenarios are presented in the next subsection.

### 6.5.2 Results

#### 6.5.2.1 Predicting number of offered resources

We use three types of time series models namely AR (autoregressive), MA (moving average) and ARIMA. In AR model, the output is regressed from the previous values. Similarly, in MA model, the output is regressed from the residual of the previous values. ARIMA combines both AR and MR. In other words, ARIMA forecasts the current output by taking previous values and residuals into account. Apart from time series models, we perform additional prediction. In the first method, the current prediction is the mean of all the previous values. In the second method, the current prediction is the immediate past value. The result of time series modeling and additional approaches of the randomly sampled CPU usage is presented in Figure 6.2. We tested the series for non-stationary of CPU usage using Dickey-Fuller test (test for finding stochastic process affecting time series statistical properties). The sampled series is stationary with 99% confidence level. We determined order (number of past data in time series) and moving average statistically using autocorrelation plots 6.1. The mean absolute error of the prediction approaches is presented in Table 6.2.

It is evident from the Table 6.2 that ARIMA outperforms other approaches. However, the immediate previous approach is not only computationally more straightforward than the rest of the approaches but also closer to ARIMA forecast. For time-sensitive applications, the immediate previous approach is an ideal candidate for predicting the number of offered

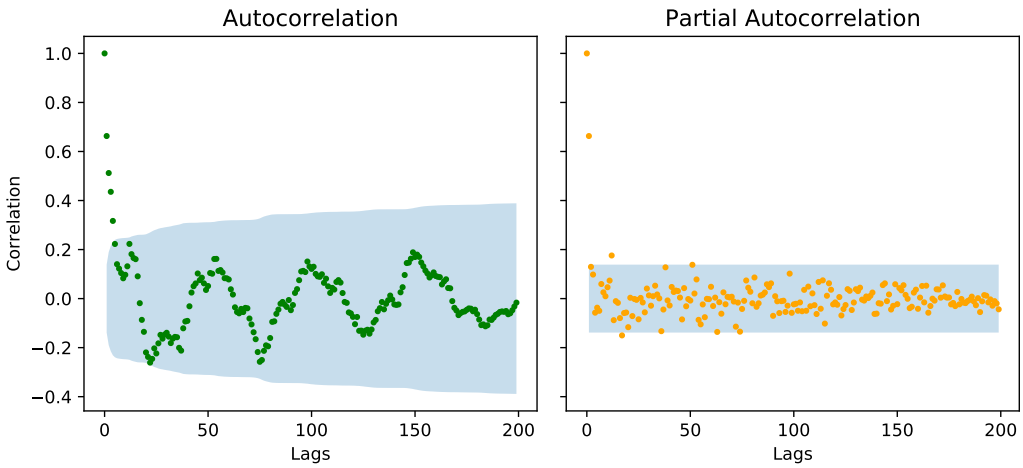



Figure 6.1: Autocorrelation and partial autocorrelation of CPU demand data.

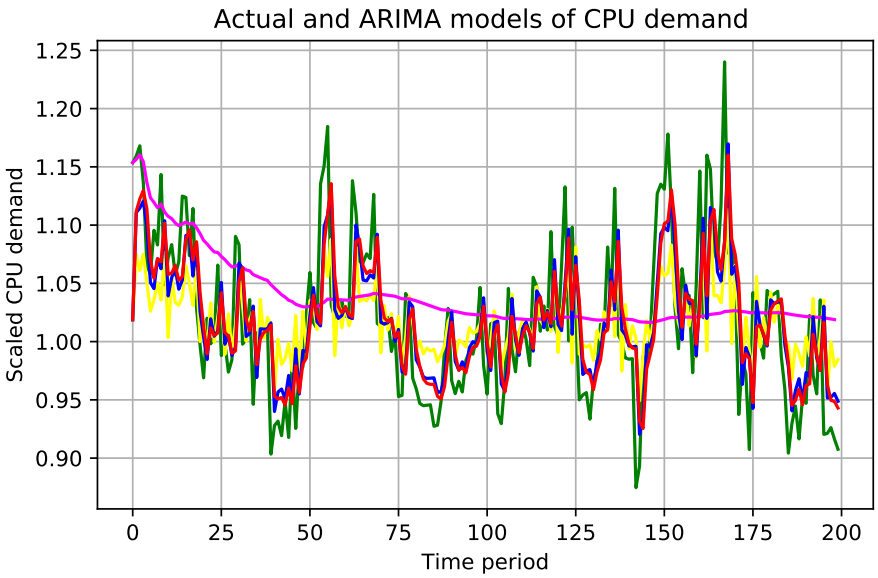



Figure 6.2: Comparison of prediction based on ARIMA, mean and previous values of sampled CPU demand of Google cluster trace

| Approach | MAE |
|---|---|
| ARIMA(1,0,1) | 0.038524 |
| AR(1,0) | 0.038602 |
| MA(0,1) | 0.044700 |
| Mean of previous | 0.053589 |
| Immediate previous | 0.039389 |

Table 6.2: Mean absolute error (MAE) for ARIMA, mean and previous values of sampled CPU demand of Google cluster trace.

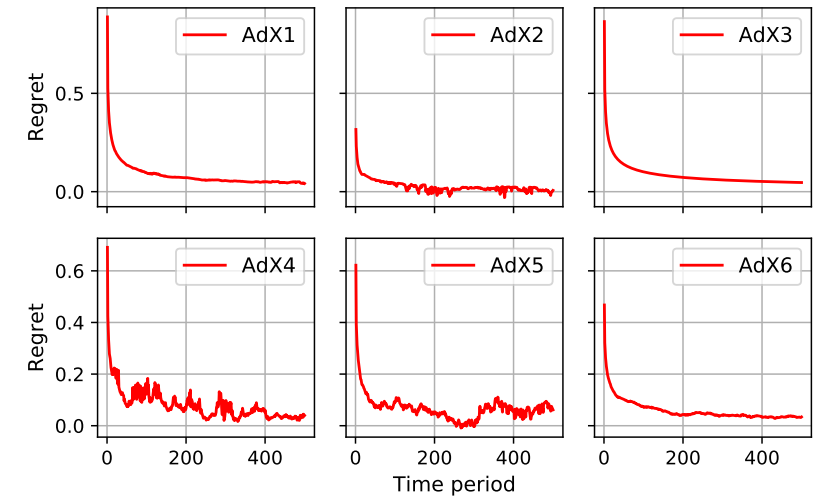
resources.

#### 6.5.2.2 Fixed Resource set

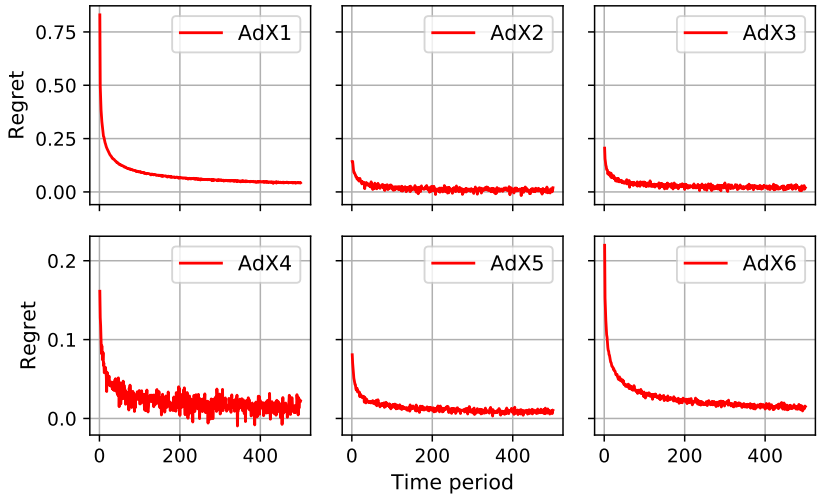
The experiment is performed for a total period $T = 500$ on both AdX dataset (Ornstein-Uhlenbeck process) and AdX augmented with both uniform and normally distributed utility. We performed normal distribution fit on the collected CPU usage trace and the estimated parameters are used to generate normally distributed utility. The regret for AdX dataset can be found in Figure 6.3a, while Figure 6.3b and 6.3c represent regret for uniformly distributed and normally distributed data respectively.

The competitive ratio for AdX dataset can be found in Figure 6.4a, while Figure 6.4b and 6.4c represent regret for uniformly distributed and normally distributed data respectively. The convergence of OFM over a period is readily evident from all the figures. Hence, the regret is reduced with time and competitive ratio increases over time. The prediction is about 95% of optimal prices on all three datasets AdX, normal and uniformly distributed data.

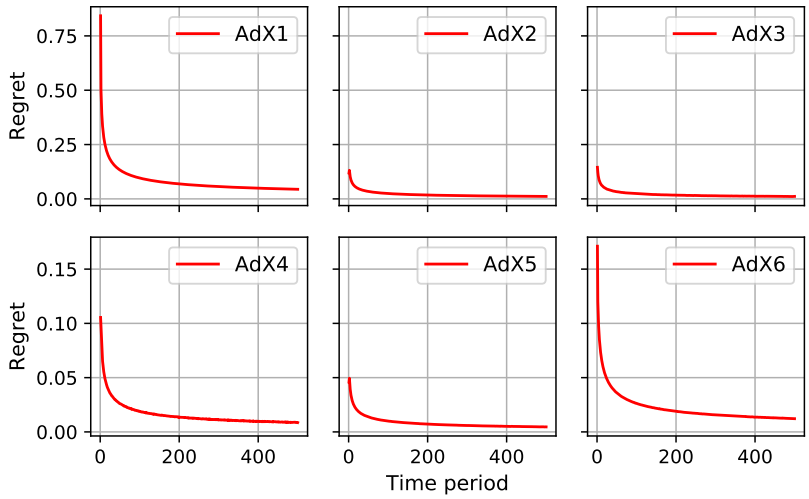
There are some cases (for instance in Figure 6.4a AdX 4) where the competitive ratio is higher than one and the corresponding regret is negative. In such cases, the value of predicted objective of OFM is higher than the optimal objection due to constraint violation of Eq. (6.2.4). In other words, the predicted prices would result in a budget violation. As soon as the input is revealed, OFM corrects itself in the next period as evident in the figures due to an absence of successive violations. The number of violations is negligible on both AdX data set and the normally distributed set. The maximum number of violations observed is 7.8% for the uniformly distributed data. In a uniform distribution, all kind of input data (best case, average case, worst case) appear with equal probability. Hence, we find a higher violation when the data is uniformly distributed.




(a) Real Adx Dataset

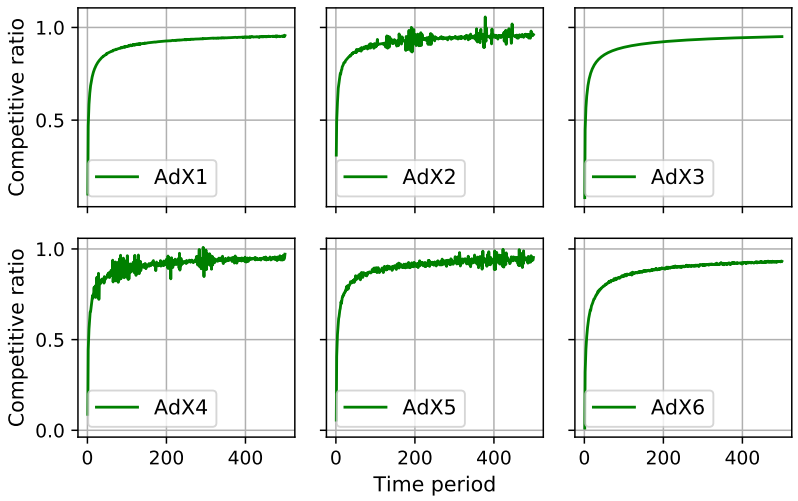



(b) Uniformly distributed

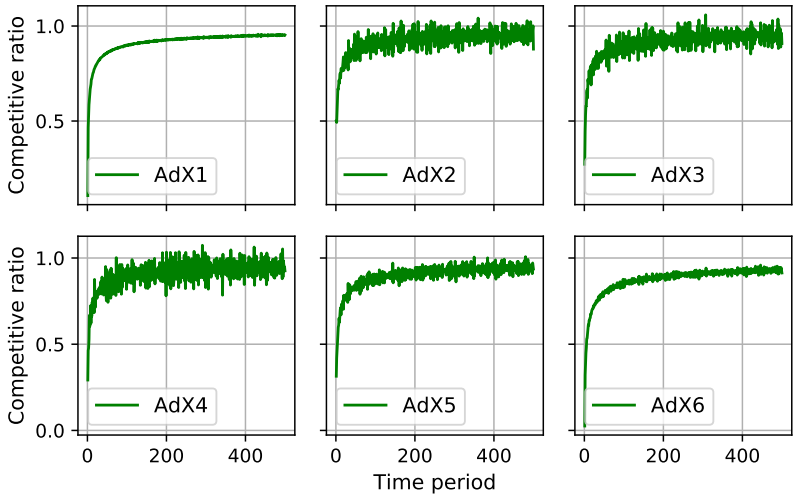



(c) Normally distributed

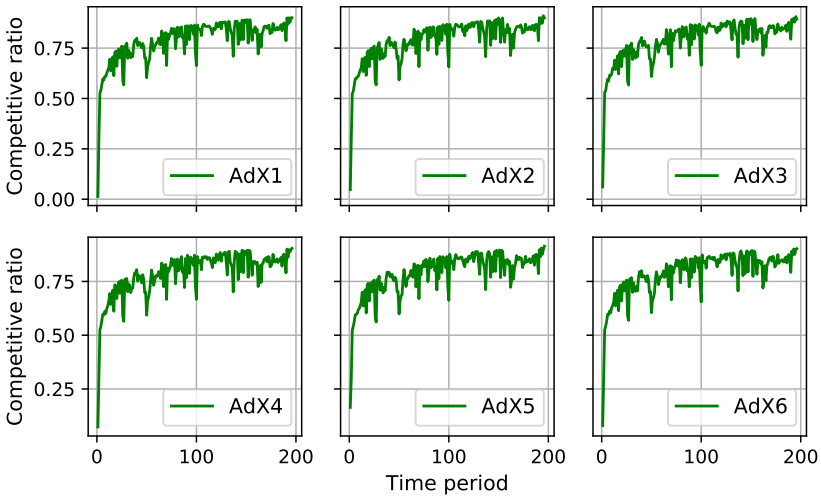
Figure 6.3: Regret for fixed resource set




(a) Real Adx Dataset




(b) Uniformly distributed




(c) Normally distributed

Figure 6.4: Competitive ratio for fixed resource set

#### 6.5.2.3 Varying Resource set

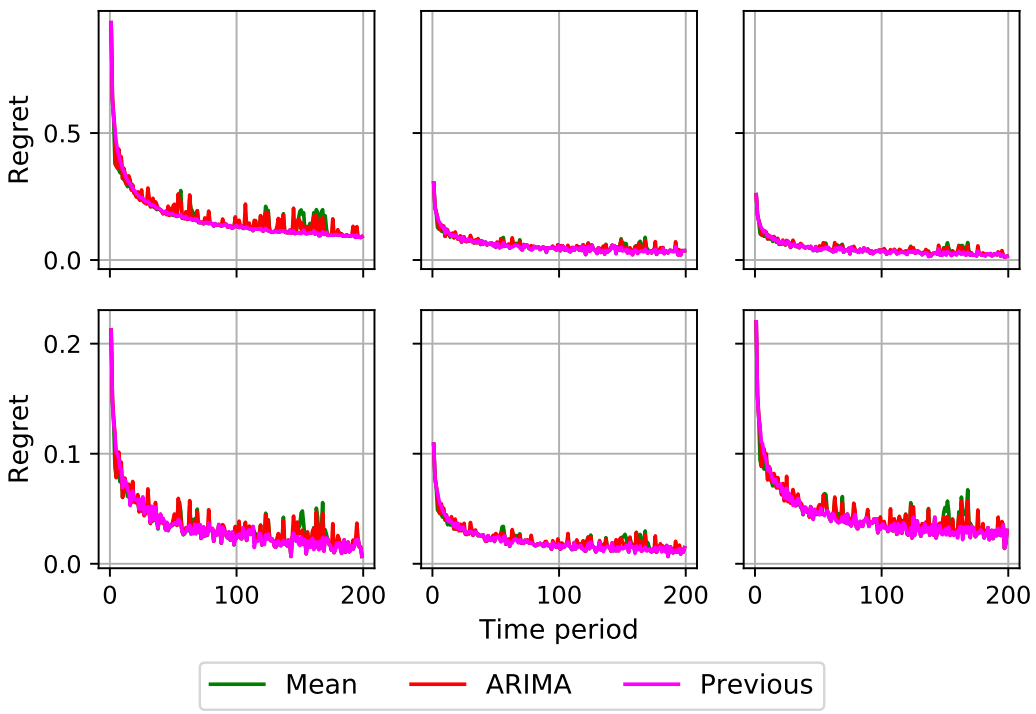
The Figure 6.5 shows the regret for uniform and normally distributed data for three prediction approaches namely ARIMA, immediate previous and mean. In this scenario, regret is not smooth and varies unlike the fixed set of resources due to the difference between the actual and predicted resource offered. We see a sudden decrease in competitive ratio, when the actual values are decreased suddenly, i.e., a smaller value in an increasing sequence.

#### 6.5.2.4 `OFM` Buyer Scalability

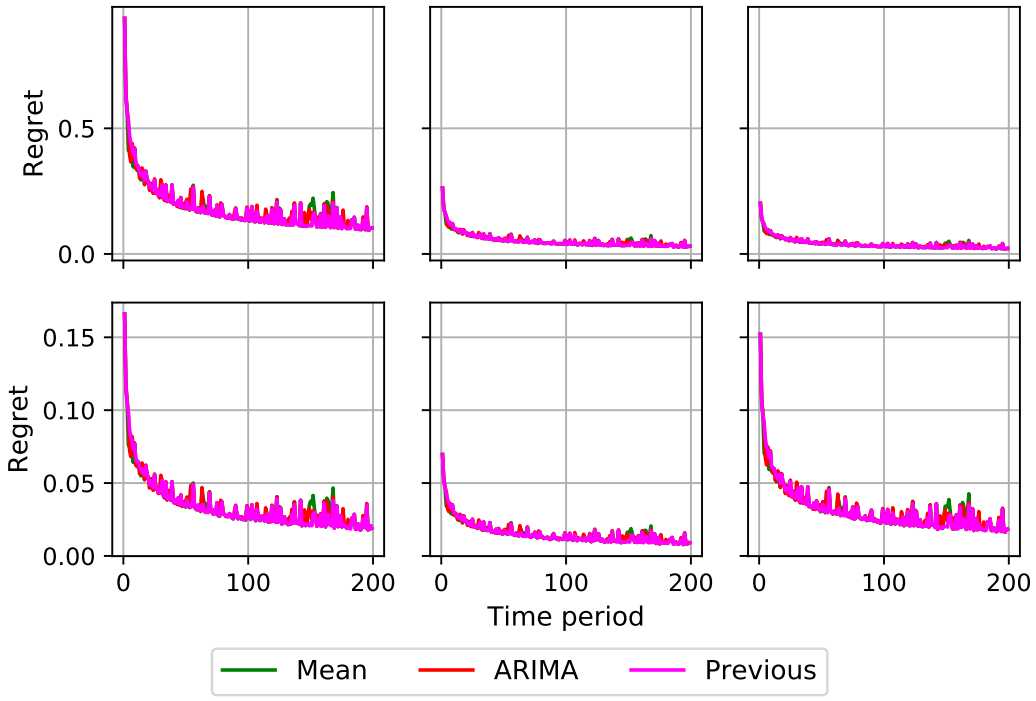
We evaluate the `OFM` scalability to handle a large number of buyers. We measure the time taken by `OFM` to compute prices using *timeit* function in Matlab. *timeit* calls a function multiple times and returns the median of actual time taken. As we know, most of the functions are vectorized for performance improvement. Hence, we fix the number of resources offered to large number 1000 for the entire period. Initially, we start with 1000 buyers and at every period, we increase the buyers by 1000. At the end of the period $t = 50$, the number of buyers is 50000. We repeat the experiment with $2000, 3000, 4000$ and 5000 as the number of resources offered. Figure 6.7 shows the time required by `OFM` to compute price. It is evident from the figure that the computational time is in the order of *microseconds* which implies that `OFM` is an apt candidate for real-time deployment.

## 6.6 Summary

In this chapter, we proposed `OFM` for computing prices for Fisher marketplace with integer allocation for varying buyers and resources at every time instance. `OFM` is an online algorithm and based on a stricter adversarial model compared to state-of-the-art solutions, i.e., the prices once computed cannot be altered. Further, prices for next time instant is predicted even before all the inputs are revealed. The experimental evaluation on both real world and emulated dataset demonstrate the low regret bound and faster convergence over the period achieving around 95% of optimal prices. The updates in `OFM` are closed-form expressions and are computationally efficient as evident from the evaluation. `OFM` computation time is in the order of *microseconds* even for a large number of buyers. Hence, `OFM` is an ideal choice for deploying online marketplace for cloud resources, especially in edge computing.

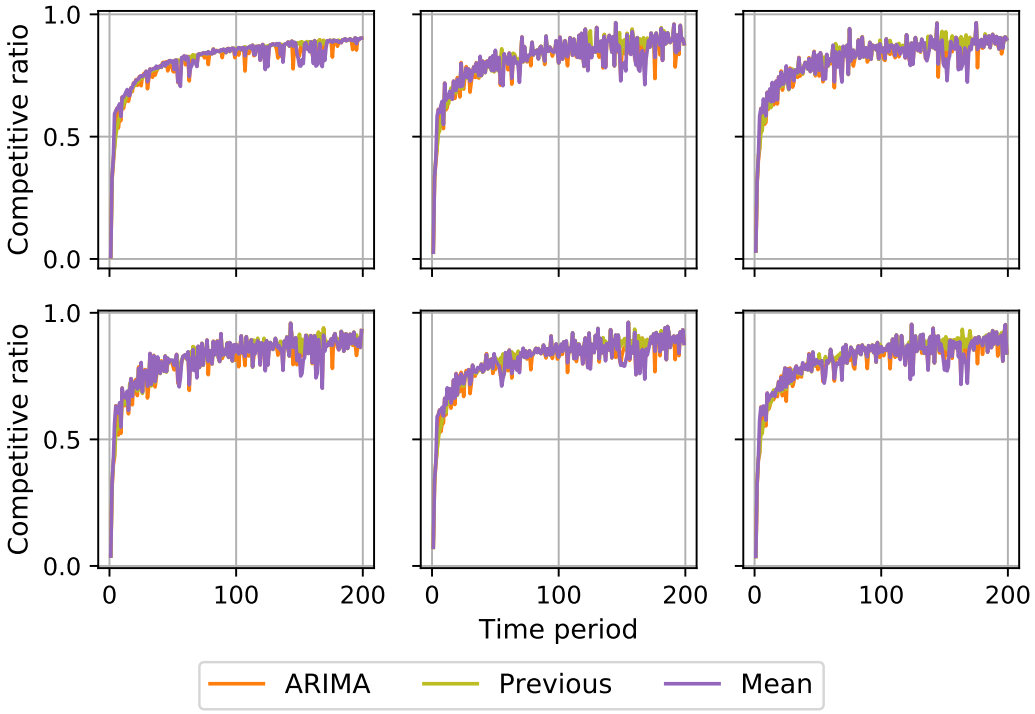



(a) Uniformly distributed

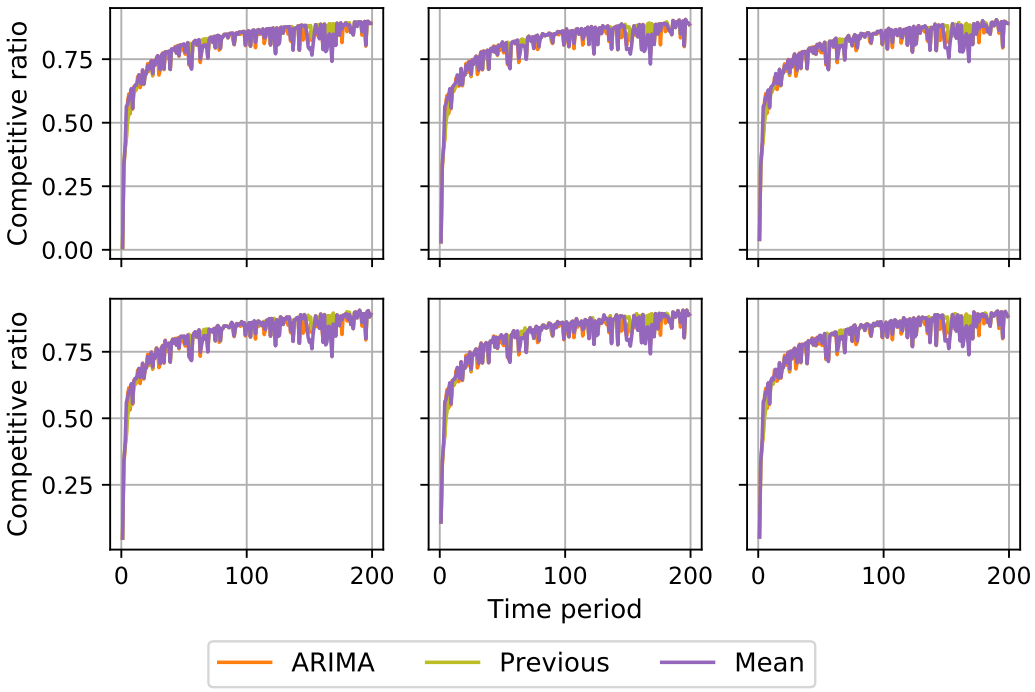



(b) Normally distributed

Figure 6.5: Regret for varying resources for ARIMA, immediate previous and mean model

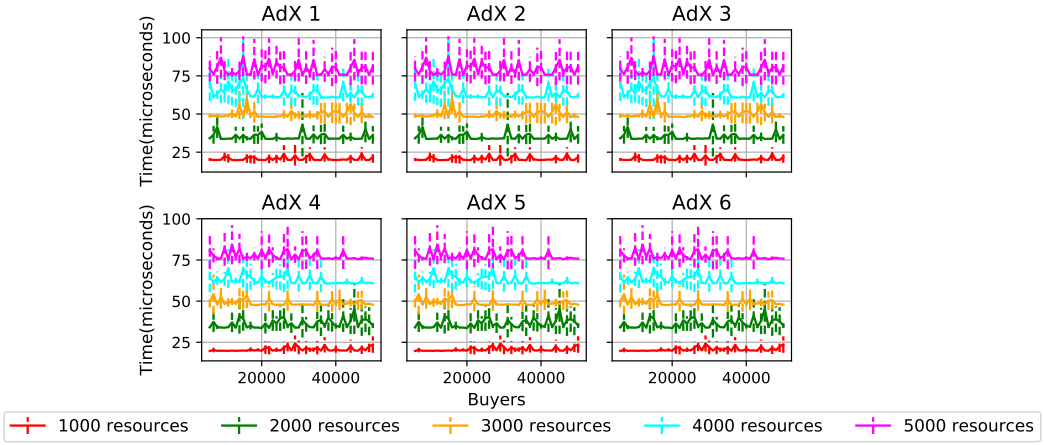



(a) Uniformly distributed

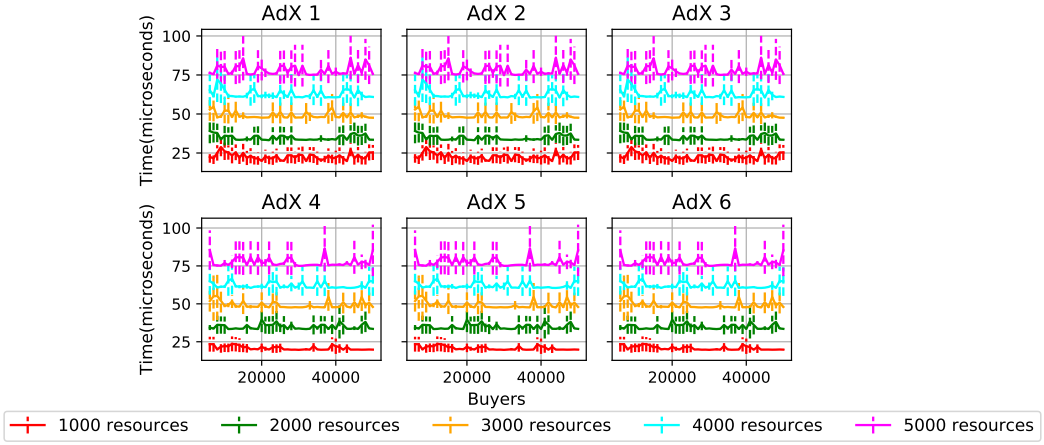



(b) Normally distributed

Figure 6.6: Competitive ratio for varying resources for ARIMA, immediate previous and mean model




(a) Uniformly distributed




(b) Normally distributed

Figure 6.7: Measured time for varying buyers for fixed resources.

# Chapter 7

# ARPP Resource Provisioning

Service providers face an important challenge of allocating resources to services optimally. Most often service providers ignore the issue of *what to provision* during deployment. This chapter proposes RConf and RConfPD as the answer to the question *what to provision*.

Initially, we motivate and formalize the problem of finding *optimal* configurations as **VCP**. Afterward, we propose a performance estimation model to estimate *latency* of a configuration based on robust queueing theory. Finally, we propose RConf and RConfPD. Furthermore, it presents the theoretical complexity and performance analysis.

## Contents

## 7.1 Introduction

One major challenge for cloud providers is resource management which typically involves monitoring of resources, provisioning resources to customers, and finally scheduling these provisioned resources. Optimizing resource management tasks frequently can yield substantial benefits in performance, revenue or energy consumption for the provider [50]. Here, one challenge in resource provisioning is optimally allocating resources to applications or services[6] requested in an ad-hoc fashion. In particular, when considering complex services, for example, multi-tier services and NFV service chains, which are composed of several distributed *components*, such as web servers, load balancers, and databases as illustrated in Figure 1.3. In the context of complex services, adequately provisioning resources while enforcing the Service Level Objectives (SLOs) expected by the customer is an arduous task.

There have been proposals, from within both industry and academia to address the problem of *when* to provision more resources (adding more component instances) for an application under high load. Proposed solutions span from threshold-based autoscaling(e.g., Google AppEngine, Microsoft Azure, Amazon AWS), to more sophisticated control theory based solutions [51–53, 144] or empirical service modeling [54, 55]. Furthermore, some proposals adopt time series analysis to make workload predictions and decide *how many* resources to provision [56,57]. These strategies are used jointly with other tools that facilitate the task of *how to* provision these resources, such as Puppet, OpenStack Heat, Ansible, and Chef.

Generally, service providers configure a requested service as a combination of various components. Furthermore, assigning a set of virtual instances and resources to each component. For instance, Network Function Virtualization (NFV) usually combines various Virtual Network Functions (VNFs) in a service chain, and assign a set of virtual instances to each VNF. Typically, each assigned instances is deployed using a single *flavor* (i.e., one VM size for a typical web server) and provisioned on demand. A component configuration can be defined through its instances, as a number of cores and an amount of memory.

However, most often service providers disregard what to provision and assign the same flavor for every instance of each component in a service, commonly known as one-size-fits-all. However, one-size-fits-all results in the following drawbacks:

- Deciding the flavor for handling incoming load requires expert knowledge of the component.
- Each flavor has different performance due to different capacities. Therefore, one unique flavor may not be suited for every amount of incoming load or type of traffic.

[6] In this chapter, term services and applications are used interchangeably.

- Evaluating a configuration in isolation of other components in the service might result in inefficient utilization or unexpected performance degradation. Generally, a one-size-fits-all solution results in deploying unneeded resources generating additional costs.

In this chapter, we address the *what to provision* problem, which we refer to as *Virtual Configuration Problem* (**VCP**). **VCP** consists of discovering the optimal flavors (using Amazon EC2 terminology) and the number of instances to be deployed for each component of a service, given a set component configurations and a workload input, while maximizing the resource utilization and ensuring the agreed SLOs requirements. Furthermore, **VCP** is reduced to *Multiple choice Multidimensional Knapsack Problem* (MMKP).

Our system offers two novel techniques to service providers for finding the answer to the *what to provision* question automatically. Furthermore, it acts as one of the fundamental building blocks to automated resource management and can be used jointly with the solutions as mentioned earlier addressing the *when* and *how to* challenges in order to build a comprehensive automated and optimized resource provisioning framework.

The first—dubbed `RConf`—requires solving a relaxed version of the integer linear programming (ILP) formulation of **VCP** and performs deterministic rounding to obtain an approximated solution to the problem. We believe that `RConf` [108] describes the first analytical approach to find the optimal configuration for multi-tier services by approaching the problem from a robust queueing theory [70] perspective.

Since the complexity of solving the ILP formulation of **VCP** can be prohibitive for more time-critical applications such as micro-services or edge applications, service providers can also opt for our second method, `RConfPD`. Here, instead of solving the ILP problem, we propose a framework based on the primal-dual approximation paradigm [67]. Intuitively, `RConfPD` settles for a slightly less optimal solution of **VCP** and can thereby provision a service faster than `RConf` by trading off solution quality for lower computational complexity. Concretely, our contributions are:

- A profiling implementation that gives accurate service rate and utilization estimations using an analytical model based on robust queueing theory. This model evaluates and estimates the impact on the service latency of various component configurations. The robust queueing theory uses uncertainty sets instead of randomization (as the standard queueing theory does), and in that way, it is possible to capture both: network traffic following arbitrary distributions; and different processing powers of various components.
- We introduce `RConf`, the first analytical solution to the *what to deploy* question in automated resource provisioning for complex services. Our evaluation, based on a

real-world deployment on Amazon EC2, shows that RConf can provide solutions for complex service deployment that yield up to 50% higher resource utilization while deploying up to 22% fewer resources than one-size-fits-all solutions. The core strength of RConf is that it can handle network traffic following arbitrary distributions as well as different resource capabilities of various components due to robust queueing theory, which uses uncertainty sets instead of randomization (as the standard queueing theory does).

- We further propose RConfPD, a scalable and time-efficient method that trades off some optimality against computation complexity. We propose an algorithm that can find an approximated solution for service provisioning based on a primal-dual framework with an approximation ratio of $\rho > 1$. We believe that this chapter describes the first primal-dual approach to find an approximate solution for MMKP problem. The basic idea is to incrementally find feasible solutions for both primal and dual formulations simultaneously. For each component, RConfPD selects the configuration with the tightest constraints after removing dominated solutions. We evaluate RConfPD analytically with MMKP benchmarks with hundreds of components and thousands of configurations for scalability [145] and a three-tiered cloud service. Our results show that while RConfPD yields a utilization close to that of the optimal solution found by RConf, it can provision even large services within $1-10$ ms and thereby one to two orders of magnitude faster.

The remainder of this chapter is structured as follows. In Section 7.2, we formally describe the problem of finding the optimal configuration for multi-tier services. Section 7.3 describes the system model that provides the input for both approaches. In Section 7.4 and 7.5, we introduce and analyze RConf and RConfPD respectively. Finally, we summarize in section 7.6.

## 7.2 VCP Problem

The provisioning of resources for a complex service can be performed in multiple ways. For instance, every instance from every component in the service can be instantiated with the same flavor. As we see from the previous section, the one-size-fits-all approach results in either over-provisioning or under-provisioning. Instead, in this chapter, we propose to leverage the usual variety of instance flavors when creating a configuration for each component of the service. Thus, the same (virtual) functionality can be provided with varying resource capacities such as CPU cores and memory. For instance, a load balancer might run in a "small" instance (e.g., 1 core, 2GB memory, and 1 Gbps bandwidth), instead of a "large" instance (e.g., 4 cores, 4GB memory) when the traffic load is low.

### 7.2.1 Numerical Example

Let us consider a hypothetical service configuration for which a customer has expressed an SLO latency requirement of $u(l) = 300ms$. In this example, our service consists of four components, a firewall (FW), a load balancer (LB), deep packet inspection (DPI) and a web server (WS). Table 7.1 shows the result of using various instance flavors (e.g., the smallest instance of an LB take $80ms$ to process the traffic, while medium instances combined achieve in $40ms$), as well as their total utilization, and the end-to-end service latency. Each row of Table 7.1 represents a feasible service configuration, that is, it satisfies the customer SLOs. The first five columns of the table represent the configuration name and the latency for the FW, LB, DPI and WS respectively, while the sixth and seventh column represents the service utilization and service latency.

While all instance combinations satisfy $u(l)$, they differ in their total utilization. Configuration A may represent the traditional way of one-fits-all (over-provisioning) approach, which meets the SLOs but results in low resource utilization. Moreover, even though both E and G exhibit the lowest latency, F represents the best configuration, as it results in a higher aggregate utilization of the components than in any other service configurations.

The challenge is then to find the combinations of configuration for each component in the service such that the complete service satisfies both service provider and customer constraints. While one particular problem may yield a large number of feasible service configurations, our goal is further to find the one single configuration of a service that maximizes the resource utilization of all service components. We define the problem of finding the optimal configuration for a complex service as the *Virtual Configuration Problem* (**VCP**).

| Config | FW | LB | DPI | WS | Utilization | Latency |
|---|---|---|---|---|---|---|
| A | 40 | 40 | 40 | 40 | 0.2 | 160 |
| B | 60 | 60 | 100 | 60 | 0.1 | 280 |
| C | 80 | 60 | 120 | 40 | 0.4 | 300 |
| D | 40 | 80 | 100 | 40 | 0.6 | 260 |
| E | 60 | 60 | 60 | 40 | 0.7 | 220 |
| F | 60 | 60 | 60 | 60 | 0.8 | 240 |
| G | 40 | 40 | 100 | 40 | 0.5 | 220 |

Table 7.1: A summary of feasible service configurations for an SLO requirement $u(l) = 300ms$. The utilization is given as $\sum_{i=1}^{n} \upsilon(\pi_i)$. All values are given in *ms* (milliseconds).

### 7.2.2 Formal Definitions

We present formal definitions for **VCP**. We denote a service composed of $n$ virtual components as $\mathscr{S} = \{v_1, v_2, \ldots, v_n\}$. The components service the flows passing through $\mathscr{S}$ in order, i.e., flows enter $\mathscr{S}$ through $v_1$ and leave through $v_n$. To capture the order in which flows are serviced we assume $\mathscr{S}$ to be a partially ordered set (poset).

A component consists of one or more instances, using the same or various flavors. We denote this set of instances as *component configuration*. Then, we define the set of feasible configurations $c$ of the $i^{th}$ component as the array $c_i = \{c_{i1}, \ldots, c_{i|c_i|}\}$. Similarly, each of these configurations can be defined as $c_{ij} = \{x_{i1}, \ldots x_{i|\theta|}\}$, where $x_{i\theta}$ represents the number of instances of a given flavor $\theta_i$, and $\theta_i$ is the set of available flavors. Each flavor $\theta_i$ has an associated tuple of $m$ dimensions describing its allocated resources (e.g., CPU cores, memory, ...). We denote the resources for the $k^{th}$ flavor as $\theta_k = \langle \theta_k^1, \ldots, \theta_k^m \rangle$. The set of configurations $c_i$ is sorted in non-decreasing order of resource requirements. Finally, we denote the set of all possible configurations for the components in a service $\mathscr{S}$ as $\mathscr{C} = \{c_i | \forall i \leq n\}$. Note that there is no restriction about the choice of instance flavors within the configuration. For instance, one configuration for a load balancer might contain multiple instances of the same type, while another configuration might contain only instances of various flavors.

Regarding resources, we denote the aggregated required resources of type $l$ within configuration $c_{ij}$ as $r_{ij}^l = \sum_{k=1}^{|\theta|} x_{jk}\theta_k^l$. The required resources are upper bounded by $u^l$ that represents the maximum resources of type $l$ available for service $\mathscr{S}$. Finally, regarding the service, we define the utilization of $c_{ij}$ as $\upsilon_{ij}$, which represents the ratio between the workload being served and the aggregated capacity of a component. Furthermore, $g_{ij}$ is defined as the SLOs of a configuration — in optimization terms, this represents the cost of a configuration. Table 7.2 summarizes the notation used in this chapter.

The optimal solution for **VCP** is that service configuration that maximizes the overall resource usage while enforcing SLOs. Before trying to find such a configuration, we need to define the set of possible configurations for a component and the entire service. Assume that we have a method that allows us to find the set of feasible configurations $c_i$ for a component, estimate their utilization $\upsilon_{ij}$, required usage $r_{ij}^l$ and cost in performance $g_{ij}$, given an input workload. Having these estimations, we can formulate **VCP** as the following integer linear

| Symbol | Description |
|---|---|
| $\mathscr{S}$ | Service |
| $v_i$ | $i^{th}$ virtual component |
| $c_i$ | Set of configurations for $v_i$ |
| $c_{ij}$ | Configuration $j$ for $v_i$ |
| $\theta$ | Set of available flavors |
| $\lambda_i$ | Traffic arrival rate at $v_i$ |
| $\mu_{ik}$ | Service rate for flavor type $\theta_k$, and $v_i$ |
| $\rho_{ik}$ | Traffic intensity for flavor type $\theta_k$, and $v_i$ |
| $\tau_{ik}$ | System time for flavor type $\theta_k$, component $i$ |
| $\mathscr{U}_i^a$, $\mathscr{U}_{ik}^s$ | Uncertainty set for arrival and departure processes |
| $\Gamma_i^a$, $\Gamma_{ik}^s$ | Arrival and departure process variability. |
| $\sigma_i^a$, $\sigma_{ik}^s$ | Standard deviation of arrival and departure processes |
| $r_{ij}^l$ | Aggregated required resources of type $l$ for $c_{ij}$ |
| $\upsilon_{ij}$ | Utilization of $c_{ij}$ |
| $x_{jk}$ | Number of instances of a given flavor $\theta_k$ for $c_{ij}$ |
| $u^l$ | Upper bound of resource type $l$ |
| $g_{ij}$ | Performance cost for $c_{ij}$ |
| $w_{ij}$ | Binary indicator of configuration selection |

Table 7.2: A list of notations used in this chapter.

program (ILP):

$$
\begin{aligned}
\underset{w}{\text{maximize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{|c_i|} v_{ij} w_{ij} \\
\text{subject to} \quad & (C1) \quad \sum_{i=1}^{n}\sum_{j=1}^{|c_i|} r_{ij}^{l} w_{ij} \leq u^{l}(\mathscr{S}), \quad \forall l = 1,2,\ldots,m, \\
& (C2) \quad \sum_{j=1}^{|c_i|} g_{ij} w_{ij} \quad \leq SLO(\mathscr{S}), \\
& (C3) \quad \sum_{j=1}^{|c_i|} w_{ij} \quad = 1, \quad \forall i = 1,2,\ldots,n, \\
& (C4) \quad w_{ij} \quad \in \{0,1\}.
\end{aligned}
\tag{7.2.1}
$$

where $w_{ij}$ is a binary indicator that represents whether a given configuration is selected or not (C4). If $w_{ij} = 1$ then configuration $c_{ij}$ is included, otherwise it is excluded. The first constraint (C1) implies that the resources required by a configuration cannot exceed any of the budgeted resources. The second constraint (C2) controls that the performance costs are not exceeded (e.g., latency). Furthermore, (C3) limits the number of selectable configurations for every component to 1.

**Theorem 7.1 (VCP complexity )** **VCP** is NP-hard.

**Proof** In a classical multiple-choice multidimensional knapsack problem (MMKP), we have $n$ mutually exclusive item classes, $|J_i|$ items per class, and multiple dimensions per item ($l$). Out of these items, we can only select one per class. Moreover, each item has an associated profit, and the goal is to maximize the total profit while respecting the knapsack constraints [146].

The reduction from MMKP to **VCP** is as follows. In **VCP**, the item classes are the various components, the items are the various component configurations ($|c_i|$), and the types of resource in a configuration are the various item dimensions. The profit in **VCP** depends on the utilization $v_{ij}$ of the configuration resources.[7] Finally, **VCP** aims to maximize the utilization of resources by selecting one configuration for each component. Therefore, as MMKP is NP-hard [146], **VCP** is also NP-hard.

[7] In the remainder of this chapter, we use the terms utilization and profit interchangeably.

## 7.3 Modeling a Complex Service

We have seen that **VCP** formulation assumes knowledge of performance cost that workload experiences when injected into a system with a particular configuration (Constraint C2). To acquire this knowledge, we need to model the latency of the different components for the different flavors. The accuracy of the service provisioning depends on the accuracy achieved when modeling the service key performance indicators (KPI) subject to SLOs. This section describes how to model latency accurately with robust queueing theory, as it is one of the significant KPIs for cloud services. Section 2.3 presents the fundamentals of robust queueing theory.

### 7.3.1 Robust queue Motivation

The basic idea is to model each combination of VM flavor $\theta_k$ and component $v_i$ as a queue $q_{ik}$ using one or multiple servers (i.e., CPU cores) with either limited or unlimited buffer capacity. Arrival processes enter the system (i.e., VM), and are either processed or wait in a queue (if the system is busy). Once a process is served and leaves the system, it is known as a departure process.

Applying traditional queueing theory to model complex services is either not possible or highly inaccurate due to the assumption that process's arrivals and departures are Markovian or deterministically distributed, contrary to many cloud or networks services [58, 147–150]. For instance, Ethernet traffic is heavy-tailed and self-similar [148].

Complex services could be modeled as networks of queues, where each component of the service is modeled as a G/G/m queue [76]. However, this translates to making the system analysis computationally intractable [70, 77]. To overcome these limitations, Bandi *et al.* [70] apply robust optimization for analyzing a single, or network of G/G/m queues, and subsequently, derive a closed-form expression for the *system time*.

Robust queue analysis does not consider arrival and departure processes to be arbitrarily distributed, as traditionally done in G/G/m queue models. Instead, robust optimization is performed on uncertainty sets (constraints are allowed to vary within this set) without affecting the optimality of the solution [71], and then determines the expected system time based on those uncertainty sets. In this chapter, we apply [70]to model the complex cloud services. Furthermore, both `RConf` and `RConfPD` use this model as a decision-making tool when evaluating different component configurations.

### 7.3.2 Service's Component Modeling

In this chapter, we model each instance flavor type of a given service's component as a robust queue $q_{ik}$. We assume that each flavor $\theta_k$ is duple of assigned CPU cores ($\theta_k^1$) and memory ($\theta_k^2$). Our goal is to obtain an estimation of the latency of a component configuration $c_{ij}$ given CPU cores, memory and load.

The system time for $q_{ik}$ is calculated for network traffic flow instead of traffic request. Limiting the solution to a specific request type, especially in the application layer, would affect the applicability of the solution [59]. For instance, in a typical data center, there are hundreds or thousands of different services. Therefore, it would be beneficial not to model each request of each service as a unique process, but instead to monitor the transport layer, where one can capture traffic either at *packet* or *flow* level. Packets give more information compared to flows, but require significant processing overhead, especially in environments where the number of packets grows exponentially (e.g., data centers). Robust queue models require only the arrival and departure times of the requests, which can be obtained at the flow level.

Flows enter a given component with an arrival rate $\lambda_i$ and are processed at rate $\mu_{ik}$ (service rate). If there are available servers (CPU cores) in the system, the flows are processed, otherwise flows have to wait. Using standard definitions from queueing theory [76], we define $\rho_{ik}$ as the utilization of an instance with flavor $k$ for the $i^{th}$ component:

$$\rho_{ik} = \frac{\lambda_i}{\theta_k^1 \mu_{ik}} \tag{7.3.1}$$

where $\lambda_i$ is the arrival rate of component $i$, $\mu_{ik}$ the service rate and $\theta_k^1$ the total number of CPU cores of an instance $\theta_k$.

Moreover, as indicated in chapter 2.3, robust queue models require the construction of uncertainty sets for both arrival and departure processes. Before presenting the closed form expression for the system time, it is necessary to construct these sets. The uncertainty sets are based on a stable distribution. According to the central limit theorem, any non-heavy tailed distribution converges to a normal distribution as $n \to \infty$ [70]. The normal distribution is a special case of the stable distribution [70, 151]. Nolan [151] proves that by setting the tail coefficient $\alpha = 2$, the stable distribution behaves like a normal distribution. In this way, a non-heavy tailed distribution can be converted to a stable distribution. Heavy-tailed distributions are those whose tails are not exponentially bounded like in Pareto or log-normal distributions (e.g., Ethernet traffic is heavy-tailed and self-similar [148]). In our model, heavy-tailed distributions are converted to stable distributions by setting the tail coefficient $\alpha$ to appropriate values using Hill's estimator [152] with order statistics

approximately $< 0.13\%$ of sample size [153]. Hence, our model can handle both heavy-tailed and non-heavy tailed traffic.

#### 7.3.2.1 Uncertainty set for arrivals

In this chapter, robust queue models are created per flavor type $\theta_k$ and component $v_i$. Moreover, we assume that the arrival traffic requests are the same for all the instances. Therefore, at least for arrivals, one uncertainty set is sufficient.The uncertainty set for inter arrivals of component $i$ is denoted by:

$$\mathscr{U}_i^a = \left\{ (T_{i,1}^a, T_{i,2}^a, \dots T_{i,t}^a) \middle| \frac{\sum_{j=t'+1}^{t} T_{i,j}^a - \frac{(t-t')}{\lambda_i}}{(t-t')^{\frac{1}{\alpha_i}}} \geq -\Gamma_i^a, \quad \forall 0 \leq t' \leq t-1 \right\} \tag{7.3.2}$$

where $T_{i,1}^a, T_{i,2}^a, \dots T_{i,t}^a$ denotes the inter-arrival times of the flows at component $i$ with an expected arrival rate $\frac{1}{\lambda_i}$. Moreover, $t-t'$ is the number of arrivals considered while constructing the uncertainty set. If $t'$ is equal to 0, it means that all arrivals are considered for uncertainty set construction. In our case, we set $t'=0$ to validate the robust queue model with measured data. In the above equation, $\Gamma_i^a$ captures the variability in the inter-arrival times. Since the standard deviation measures the spread of data around the mean, we set $\Gamma_i^a$ as the standard deviation of inter-arrival times, i.e., $\Gamma_i^a = \sigma_i^a$.

#### 7.3.2.2 Uncertainty set for departures

The service rate of an instance depends on the instance attributes (i.e., number of processors, memory allocated, and SLOs requirements). Hence, the departure set is defined per instance, and the uncertainty set for $\theta_{ik}$ is denoted by:

$$\mathscr{U}_{ik}^s = \left\{ \left(T_{ik,1}^s, T_{ik,2}^s, \dots T_{ik,t}^s\right) \middle| \frac{\sum_{l=t'}^{t} T_{ik,l}^s - \frac{(t-t'+1)}{\mu_{ij}}}{(t-t'+1)^{\frac{1}{\alpha_i}}} \leq \Gamma_{ik}^s, \quad \forall 0 \leq t' \leq t \right\} \tag{7.3.3}$$

where $T_{ik,1}^s, T_{ik,2}^s, \dots T_{ik,t}^s$ denotes the service times of the flows at the $i$-th component with an expected service rate $\frac{1}{\mu_{ik}}$. Furthermore, $\Gamma_{ik}^s$ captures the variability of departure process. Since the service rate depends on instance type $\theta_k$, component utilization, arrival process variability and standard deviation of the departure process, we thereby define as

$\Gamma^s_{ik} = f(\theta_k, \rho_{ik,}\Gamma^a_i, \sigma^s_i, \alpha_i)$, and given that each system is different and the function $f$ is not known, we perform general linear regression to determine $\Gamma^s_i$.

$$\Gamma^s_{ik} = \beta_1\theta_{ik} + \beta_2\rho_{ik+}\beta_3\Gamma^a_i + \beta_4\sigma^s_i + \beta_5\alpha_i + \varepsilon \tag{7.3.4}$$

where $\varepsilon$ is the difference between measured and estimated $\Gamma^s_{ik}$. We use the closed form expression of linear regression to compute the values $\beta_1, \ldots, \beta_5$.

Then the estimated system time $\tau_{ik}$ is given by:

$$\tau_{ik=}\frac{(\alpha_i - 1)}{\alpha_i^{\frac{\alpha_i}{(\alpha_i-1)}}}\frac{\lambda_i^{\frac{1}{(\alpha_i-1)}}\Gamma^a_i + \left(\frac{\Gamma^s_{ik}}{(\theta^1_{ij})^{\frac{1}{\alpha_i}}}\right)^{\frac{\alpha_i}{(\alpha_i-1)}}}{(1-\rho_{ik})} + \frac{\theta^1_k}{\lambda_i} \tag{7.3.5}$$

Initially, we calculate output variability $\Gamma^s_{ik}$ by substituting the observed average system time in Equation (7.3.5). We also consider the standard deviation of arrivals as input variability $\Gamma^a_i$. Once we have set $\Gamma^s_{ik}$, we perform a linear regression using Equation (7.3.4) to predict the next $\Gamma^s_{ik}$ values and use these values again in Equation (7.3.5) to estimate the system time. Even though Bandi *et al.* [70] derive the system time expression for the worst case, we train the model using average flow processing times.

### 7.3.3 VCP Metrics

We now describe and formalize the metrics required for solving the **VCP** problem as introduced in Section 7.2.

**Utilization**: the utilization of $j^{th}$ configuration of an $i^{th}$ component $\upsilon_{ij}$ is defined as:

$$0 \leq \upsilon_{ij} = \frac{\sum_{k=1}^{|\theta|}\rho_{ik}x_{jk}}{\sum_{k=1}^{|\theta|}x_{jk}} \quad \leq 1, \tag{7.3.6}$$

where $x_{jk}$ represents the number of instances of $\theta_k$ for configuration $c_{ij}$. In order to compute $\rho_{ik}$, we compute the service rate $\mu_{ik}$ (Equation (7.3.1)) as follows.

In a queueing model, system parameters are derived based on arrival and departure time epoch, i.e., $T^s_{ik,t}, T^a_{i,t}$. Hence, only the system time is observed. The system time is the sum of waiting time and processing time. The service rate measures how fast a server serving requests. Therefore, we compute the processing time from system time based on *Lindley's*

*recursion* [154] which represents the waiting time of a current request as a recursive relation between system and processing times of the arrival and departure processes. Conversely, it is non-trivial in the case of $m$ multiserver queues since $m$ servers are servicing in parallel and anyone among them can service a flow. Krivulin [155] extends the idea of *Lindley's recursion* to a G/G/m and derives a recursive relationship between the system, waiting and processing time. The basic idea of Kirvulin is to sort the departure time epochs and then compute the difference between arrival and departure time epochs. Also, in an $m$ server queue, the first $m$ departure processes have zero waiting time. Hence, once we know the processing time $T^s_{ik,t}$ we can compute $\mu_{ik} = \frac{1}{\mathbb{E}(X_{ik,t})}$.

**Resource usage**: In our case, the resource requirements function is trivial for the number of cores and memory, $r^l_{ij} = \sum_{k=1}^{|c_i|} \theta^l_k x_{jk}$, for l=1,2.

**Performance cost**: Our performance metric for a configuration is *latency*. We compute it using Equation (7.3.5). Hence, $g_{ij} = \max_{\forall k \in \theta_k} \tau_{ik}$. Intuitively, in a scenario with multiple instances, the traffic is split between component instances since different instances work in parallel. Therefore, we define the latency of a component as that of the instance with the highest system time.

**Capacity**: We define the capacity as the quantity of available resources or type $l$, i.e., cores or memory. We denote it as $u^l$

**SLO**: As we use latency as our performance metric, our SLO is the maximum acceptable latency for the service.

## 7.4 RConf

In this section, we present an approximation algorithm for **VCP** based on the relaxation of the formulation presented in Section 7.2, and a deterministic rounding process. Furthermore, we analyze complexity and performance of `RConf`.

### 7.4.1 Algorithm

Algorithm 7.1 presents the pseudo-code for **VCP**. The algorithm takes an expected workload $\lambda$ as input and computes the set of configurations $\mathscr{C}$ that can serve it. From $\mathscr{C}$ we can compute the set of resources of each type $l$ required by each $c_{ij}$, $r^l_{ij}$, its utilization $\upsilon_{ij}$, and

its performance cost $g_{ij}$. The set $\mathscr{C}$ can be trivially computed with greedy approaches. For instance, for each $v_i$, take the minimum number of instances of the flavor with the fewer resources as the first configuration. From there, we start replacing a number of these instances by equivalent superior instances iteratively. A configuration is finally given by a set of instances, which is sufficient to serve $\lambda$ but that would not be if we remove any of the selected instances. Knowing $\mathscr{C}$, their resource requirements are computed as $r^l_{ij} = \sum_{k=1}^{|\theta|} x_{jk}\theta^l_k$.

We start by computing the set of initial parameters (line 1). Then, we relax condition C4, converting our ILP into a linear problem. Using the formulation from Equation 7.2.1 with solvers such as CPLEX, we can obtain a fractional optimal configuration, obtaining the set of coefficients $w_{ij}$ (line 2). In case of a solution, we initialize variables $P$ and $R$ to store our intermediate rounding steps. Otherwise, the algorithm terminates (line 4).

`RConf` first evaluates the trivial rounding making the highest coefficient for each component equal to 1 and zeroing the rest (lines 6-8). This configuration is returned if it meets C1 and C2. If not, `RConf` tackles them alternatively in an iterative process. First, if performance constraints are not satisfied (C2), we select the component contributing the most to the performance costs and try to select a configuration with more resources to reduce its cost. As $c_i$ is sorted in non-decreasing order of resources, we choose the highest coefficient between the current and the configuration with most resources for that component, make it 1 and zero the rest (lines 11-13). Before updating the coefficients $w'_{ij}$, we check if the selected component has not been previously pushed in the other direction, i.e., we have not reduced its resources. If we have not, it is absent in set $R$. Similarly, we also check $P[i']$ to see whether we have already increased the resources of this component configuration, but not as much as now. If the condition is satisfied, we update the coefficients $w'_{ij}$ and add the duple $(i', j')$ to $P$. If not satisfied, we try with the next component regarding the performance cost. If no component can be updated, `RConf` fails to find a valid configuration in terms of performance constraint (lines 14-20).

Then, we evaluate constraint C1. If it is not satisfied, we follow a similar procedure, but this time taking the component consuming the most resources (line 22). We try to switch to a configuration with fewer resources by choosing the largest coefficient between $w_{i1}$ and our current $w'_{ij}$, making it 1 and zeroing the others (lines 23-24). We then check whether this component has been updated before or not by checking sets P and R. If conditions are satisfied, we update $w'_{ij}$ and add the duple $(i', j')$ to $R$ (lines 25-27). Otherwise, we try with the next component regarding resource requirements (line 29) or fail to return a configuration (line 30). When both constraints C1 and C2 are satisfied, `RConf` returns the current $w'_{ij}$ that indicates the required configuration for each component (line 31).

It is worth noting that the models obtained following the robust queue based models are

**Algorithm 7.1** `RConf` algorithm.

**Require:** $\lambda$

$\{\mathscr{C}, \sum_{l=1}^{m} r_{ij}^{l}, \upsilon_{ij}, g_{ij}\} \leftarrow$ get_init_params($\lambda$)
$w_{ij} \leftarrow$ solution of relaxed **VCP**
**if** No valid $w_{ij}$ **then**
    exit()
$P, R \leftarrow \emptyset$
**for** $i \leq n$ **do**
    $w'_{ij'} \leftarrow 1$ for $j' : j' : w_{ij'} \geq w_{ij} \; \forall \; j \neq j'$
    $w'_{ij} \leftarrow 0$
**while** C1 or C2 are not met **do**
    **if** C2 not met **then**
        $i' : w'_{i'j} \cdot g_{i'j} > w'_{ij} \cdot g_{ij}, \; \forall \; i \neq i', j$
        $j'' : w'_{i'j''} == 1$
        $j' : w_{i'j'} > w_{ij}$, for $j'' < j' \leq |c'_i|$
        **if** $i' \notin R$ or $(i' \in P$ *and* $P[i'] < j')$ **then**
            $w'_{i'j'} \leftarrow 1, \; w'_{i \neq i', j \neq j'} \leftarrow 0$
            $P[i'] = j'$
        **else if** $w'_{i'j} \cdot g_{i'j} \not< min(w'_{ij}) \cdot g_{ij})$ **then**
            try next $i'$
        **else**
            No valid configuration found; Exit
    **if** C1 not met **then**
        $i' : w'_{i'j} \cdot (r^{k}_{i'j} / \upsilon^{k}) \geq w'_{ij} \cdot (r^{k}_{ij} / \upsilon^{k}), \; \forall \; i \neq i', j, k$
        $j'' : w'_{i'j''} == 1$
        $j' : w_{i'j'} > w_{ij}$, for $0 \leq j' < j''$
        **if** $i' \notin P$ or $(i' \in R$ *and* $R[i'] > j')$ **then**
            $w'_{i'j'} \leftarrow 1, \; w'_{i \neq i', j \neq j'} \leftarrow 0$
            $R[i'] = j'$
        **else if** $w'_{i'j} \cdot (r^{k}_{i'j} / \upsilon^{k}) \not< min(w'_{ij}) \cdot g_{ij})$ **then**
            try next $i'$
        **else**
            No valid configuration found; exit()
Return $w'_{ij}$

crucial for the performance of RConf. Inaccurate models would lead to over or under-provisioning, lowering the resource utilization or degrading service performance respectively. Similarly, RConf's primary asset is the rounding process performed over the solution provided by linear solvers (CPLEX in our case) to the relaxed version of **VCP**. That initial solution is not implementable (we cannot deploy fractions of instances) and applying a trivial rounding will usually not meet both C1 and C2 constraints, requiring a more elaborate solution like the one proposed. RConf can find near-optimal solutions. However, this comes at a price of complexity. To quantify this trade-off, we present a performance and complexity analysis of RConf in this section.

### 7.4.2 Complexity and Performance analysis

At its core, RConf solves a relaxed linear problem using a linear solver (e.g., CPLEX) applying the simplex algorithm [156]. For problems with $n$ variables and $m$ constraints, simplex finds an optimal solution in polynomial time (on average), or in the best case in $\mathcal{O}(m \cdot \log m)$ [156]. However, in problems with a large number of variables, it is more appropriate to assume $\mathcal{O}(n^m)$ [157], the simplex worst-case complexity.

**VCP** has $\sum_{i=1}^{n} |c_i|$ variables, where $|c_i|$ are the configurations per component, and $n$ is the number of components. The number of constraints is $m+1$, $m$ different resources plus one for performance. In production environments, we may need to choose among a large number of feasible configurations for multiple components as well as consider multiple resources. Therefore, we assume a simplex worst-case complexity in our analysis. The complexity for the linear programming part is $\mathcal{O}\big(\sum_{i=1}^{n} |c_i|\big)^{m+1}$. The posterior iterative rounding process can take up to $\mathcal{O}\big(\sum_{i=1}^{n} |c_i|\big)$ in the worst case. Combining both components, RConf has a worst-case time complexity of $\mathcal{O}\Big(\big(\sum_{i=1}^{n} |c_i|\big)^{m+1} + \big(\sum_{i=1}^{n} |c_i|\big)\Big)$.

We now show the approximation ratio of RConf with Theorem 7.2.

**Theorem 7.2 (RConf approximation ratio)** RConf has $1+\varepsilon$ approximation ratio.

**Proof** Let us define $\boldsymbol{Z^{\star}_{VCP}}$ as the aggregated resource utilization of the optimal solution for an instance of **VCP**(Eq. 7.2.1). The decision variable $w_{ij} = \{0,1\}$ represents whether the optimal solution selects configuration $j$ for component $i$. We also define $\overline{w_{ij}} = \{0,1\}$ to denote the configuration selected for each component after rounding the solution to the relaxed instance of **VCP**. Then, we have that

$$\boldsymbol{Z}^{\star}_{\boldsymbol{VCP}} = \sum_{i=1}^{n} \upsilon_{ij} \cdot w_{ij} = \sum_{i=1}^{n} \upsilon_{ij} \cdot \overline{w_{ij}} + \sum_{i=1}^{n} \delta_i.$$

The optimal resource utilization is larger or equal than that of the rounded solution usage. The term $\delta_i$ captures the difference across component resources utilization. Note that, hence, $\sum_{i=1}^{n} \delta_i \geq 0$. Additionally, the result obtained by rounding the relaxed instance solution may lead to a global configuration that violates the resource or performance contraints. `RConf` iterates over the solution replacing some of the selected configurations by other with less resources if the budget was exceeded; or more if performance constraints were exceeded. This results in

$$\boldsymbol{Z}^{\star}_{\boldsymbol{VCP}} = \sum_{i=1}^{\psi} \upsilon_{il} \cdot \widehat{w_{ij}} + \sum_{i=\psi+1}^{n} \upsilon_{i\ell} \cdot \overline{w_{ij}} + \sum_{i=1}^{\psi} \eta_i + \sum_{i=1}^{n} \delta_i.$$

Here, $\psi$ is the set of components whose configuration changed due to constraint violations. The new configuration is denoted by $\widehat{w_{ij}}$. These $\psi$ components also change their contribution to the total utilization. This variation is captured per component by the term $\eta_i$. Note that, differently to $\delta_i$, $\eta_i$ is negative when resource budget constraints are violated (new configuration has less resources, higher utilization). When performance constraints are violated $\eta_i \geq 0$ (new configurations has more resources, lower utilization).

Let $\boldsymbol{Z}^{RConf}_{\boldsymbol{VCP}}$ be the final total utilization of `RConf` and $\Delta = \sum_{i=1}^{\psi} \eta_i + \sum_{i=1}^{n} \delta_i$. Observe that, necessarily, $\Delta \geq 0$. Therefore

$$\boldsymbol{Z}^{\star}_{\boldsymbol{VCP}} = \boldsymbol{Z}^{RConf}_{\boldsymbol{VCP}} + \Delta \implies \frac{\boldsymbol{Z}^{\star}_{\boldsymbol{VCP}}}{\boldsymbol{Z}^{RConf}_{\boldsymbol{VCP}}} = 1 + \frac{\Delta}{\boldsymbol{Z}^{RConf}_{\boldsymbol{VCP}}} = 1 + \varepsilon$$

where $\varepsilon = \Delta / \boldsymbol{Z}^{RConf}_{\boldsymbol{VCP}} \geq 0$. If no constraints are violated after the rounding, $\psi = 0$ and $\sum_{i=1}^{\psi} \eta_i = 0$. This expression also captures the case where the linear solver returns the optimal solution. In that case, $\delta_i = 0 \; \forall i$ and, inherently, $\eta_i = 0 \; \forall i$, leading to $\varepsilon = 0$.

## 7.5 `RConfPD`

`RConf` computational complexity grows exponentially with problem size. Additionally, requires running a linear solver on every execution. Therefore, in practice, `RConf` can be

used for services such as web hosting, data analytics, among others, but it is not suitable for many networking or cloud services (e.g., edge computing applications or high-sensitive micro-services) which require near-instant provisioning [158]. We now introduce RConfPD, a primal-dual algorithm to find near-optimal approximations for resource provisioning with execution times substantially lower than RConf.

Section 2.1 presents the fundamentals of duality theory prerequisite for this section. The intuition behind the primal-dual approximation is to relax the complementary slackness (with the weak duality theorem) and to find a primal feasible solution starting from a feasible dual solution [67].

### 7.5.1 RConf Primal-dual and complementary slackness formulation

We presented the **VCP** ILP in Section 7.2. For RConfPD we use its relaxed version which eliminates the integrality constraint (C4), allowing $0 \leq w_{ij} \leq 1$. Next, we associate the dual variables $\pi_l$, $\psi$ and $\zeta_i$ with the constraints (C1), (C2) and (C3) of Eq. (7.2.1), respectively. Then, the corresponding linear program (i.e., the *dual* LP) is given as

$$\begin{aligned} \underset{\pi,\zeta,\psi}{\text{minimize}} \quad & \sum_{l=1}^{m} \pi^l u^l + \sum_{i=1}^{n} \zeta_i + SLO(\mathscr{S})\psi \\ \text{subject to} \quad (D1) \quad & \sum_{l=1}^{m} \pi^l r_{ij}^l + g_{ij}\psi + \zeta_i \geq \upsilon_{ij}, \quad \forall i = 1,2,\ldots,n, j = 1,2,\ldots,|c_i|. \end{aligned} \tag{7.5.1}$$

According to theorem 2.2, at optimality, both primal and dual satisfy *complementary slackness* [65]. In a primal-dual approximation, either primal or dual complementary slackness is relaxed to construct a feasible solution. For solving **VCP**, we relax the primal complementary slackness. Hence, the complementary slackness definitions for **VCP** are:

(i) $\forall i \in N, j \in c_i \implies \sum_{i=1}^{n} \sum_{j=1}^{|c_i|} r_{ij}^l w_{ij} = u^l(\mathscr{S})$ and $\sum_{i=1}^{n} \sum_{j=1}^{|c_i|} g_{ij} w_{ij} = SLO(\mathscr{S})$.

(ii) $\sum_{l=1}^{m} \pi^l r_{ij}^l + g_{ij}\psi + \zeta_i \geq \dfrac{\upsilon_{ij}}{\rho}$ such that $\rho > 1$.

For **VCP**, the dual slackness implies that violation of the resource or SLO constraints, which in turn would increase the primal profit, or the resource utilization. We therefore keep dual slackness tight. On the other hand, the (now relaxed) primal slackness implies that RConfPD is allowed to settle for a slightly reduced profit (at least $\dfrac{\upsilon_{ij}}{\rho}$) if it does not use additional resources.

### 7.5.2 `RConfPD` Algorithm

In MMKP-Knapsack problems, the concept of dominance plays an important role [159]. Consider two items $p$ and $q$ with weights $w_p$ and $w_q$, respectively. Let $\mathscr{P}_p$ and $\mathscr{P}_q$ be the corresponding profits. If $\mathscr{P}_p > \mathscr{P}_q$ and $w_p < w_q$, then item $q$ is *dominated* by item $p$, i.e., it offers greater profit at lesser weight [159]. In this case, we can remove the dominated item to save computation time without affecting the solution. Geometrically, the non-dominated (i.e., dominating) items are extreme points while dominated items are interior points of the polyhedron [146]. Since optimal solutions are located at the extreme points of the polyhedron [65], we can reduce the complexity of `RConfPD` without affecting the optimality by safely removing dominated items during a preprocessing stage [159].

Thus, removing dominated configurations requires to accurately determine their respective weights (i.e., resource consumption). In MMKP, each item is a multidimensional vector. One option to reduce this multi-dimensionality into a single dimension is to *norm* the vector [145]. In `RConfPD`, we use a similar idea and perform scaling of each resource based on its limits. We call it the *resource norm*. Consider two flavors $\mathscr{F}_1$ and $\mathscr{F}_2$ with resource requirements as shown in Table 7.3 and resource limits of 20 cores and 30GB memory (the *norms* for both flavors are also in the table). $\mathscr{F}_1$ uses a much smaller fraction of memory compared to $\mathscr{F}_2$. Therefore, scaling resources with the *resource norm* allow for a more precise estimation of resource consumption.

| Flavor | vCPU | Mem (GiB) | *norm* | *resource norm* | *Max. vCPU* | *Max. RAM* |
|---|---|---|---|---|---|---|
| $\mathscr{F}_1$ | 10 | 1 | 10.04 | 0.5011 | 20 | 30 |
| $\mathscr{F}_2$ | 1 | 10 | 10.04 | 0.337 | | |

Table 7.3: A numerical example of resource norm scaling.

We now describe `RConfPD` in detail. We provide its pseudo-code in Algorithm 7.2. First, we initialize both primal and dual variables of Eq.(7.2.1) and Eq.(7.5.1). In most primal-dual algorithms, the dual variables are increased based on some predefined rate. As Knapsack is a variant of the packing problem, we increase the dual variables based on the remaining capacity (Alg. 7.2, lines 1-5).

The main loop in `RConfPD` iterates over the different components in the service. Each of these iterations has two differentiated steps. The first step is presented as a different algorithm, Algorithm 7.3, that aims to eliminate dominated configurations. Here, we first compute the resource norm for the resource requirement of each configuration (Alg. 7.3, line 1). To compute this norm, we make use again of the robust queue based models presented in Section 7.3, as we require the performance cost for each configuration. Again,

**Algorithm 7.2** RConfPD

**Require:** $\lambda$

$limit_l \leftarrow u^l, \forall l \in 1,2,\ldots,m$
$lim_g \leftarrow SLO(S)$
$\psi \leftarrow 0$
$\forall l = 1,2,\cdots,m, \pi^l \leftarrow 0$
$\forall i \in N, j \in |c_i|, w_{ij} \leftarrow 0$
**for** $i \leftarrow 1$ to $n$ **do**
    Remove dominated configurations for $i^{th}$ component using Algorithm 7.3
    $winner \leftarrow \arg\max_{j \in c_i} \max_{j \in c_i} \frac{\upsilon_{ij}}{\|r_{ij}\|}$
    $\zeta_i \leftarrow \min_{j \in c_i} \upsilon_{ij}$
    $\forall l = 1,2,\cdots,m, \pi^l \leftarrow \frac{1}{limit_l}$
    $\psi \leftarrow \frac{1}{lim_g}$
    $\forall i \in N, \zeta_i \leftarrow \min \upsilon_{ij}, \forall j \in |c_i|$
    **while** constraint $(D1)$ is tight in Eq.(7.5.1) **do**
        Increase $\pi^l$, $\psi$ and $\zeta_i$
    $w_{ij} \leftarrow 1$
    $limit_l \leftarrow limit_l - r_{ij}^l, \forall l = 1,\ldots,m$
    $lim_g \leftarrow lim_g - g_{ij}$
**return** $w_{ij}$

**Algorithm 7.3** Dominated configuration removal for $i^{th}$ component

**Require:** $c_i, \forall i \in N$

Compute $\|r_{ij}\| = \sqrt{\sum_{l=1}^{m} \frac{r_{ij}^2}{u^l} + \frac{g_{ij}^2}{SLO}}, \forall j \in c_i, i \in N$
**for** $p \leftarrow 1$ to $|c_i|$ **do**
    **for** $q \leftarrow p+1$ to $|c_i|$ **do**
        **if** $(\|r_i p\| < \|r_i q\|)$ and $(\upsilon_{ip} > \upsilon_{iq})$ **then**
            $c_i \leftarrow c_i \setminus \{q\}$
Return non-dominated configurations

using imprecise models would invalidate the outputs of the algorithm. Afterward, we compare all configurations to check for domination relations. If a configuration $q$ has higher resource norm and lesser profit than a configuration $p$, then $q$ is dominated and removed from the configuration set for the $i^{th}$ component, (Alg. 7.3, lines 4-5). Back to the main loop in Alg. 7.2, we select a *winner* from the remaining configurations. This winner is the configuration that yields the maximum profit per unit resource norm (a.k.a. profit-to-weight ratio in a 0-1 knapsack problem)(Alg. 7.2, line 8). Then, we increase the dual variables until the constraint $(D1)$ becomes tight in Eq.(7.5.1)(Alg. 7.2, line 13). We set the corresponding primal variable of that configuration to 1 (Alg. 7.2, line 15). Finally, we update the limits based on the resource usage of the selected configuration (Alg. 7.2, lines 16 and 17). Once we have gone through all the components, we return the final service configuration.

### 7.5.3 Complexity and Performance Analysis

The removal of dominant configurations takes $\mathcal{O}(|c_i|\log|c_i|)$ steps due to sorting. Arranging configurations in non-increasing profit-to-weight ratio, we can determine the maximum and tighten constraint in $\mathcal{O}(1)$ steps. Hence, time complexity is $\mathcal{O}(|c_i|\log|c_i|)$ per component and of $\mathcal{O}\big(\sum_{i=1}^{n}|c_i|\log|c_i|\big)$ for $n$ components.

`RConf` finds an optimal solution for relaxed Eq.(7.2.1), which is a primal linear program. Therefore, by the duality theorem, Eq. (7.5.1) has a feasible solution [65]. Shamir and Rawitz [160] prove that the linear programming formulation of MMKP has a polynomial approximation time scheme (PTAS) for fixed $m$ resource dimensions. As `RConfPD` does not modify resource dimensions of the flavors, it is also a PTAS. We now study its approximation ratio.

**Theorem 7.3 (`RConfPD` approximation ratio)** `RConfPD` has an approximation ratio $\rho$.

**Proof** We denote the total utilization of an optimal solution for Eq.(7.2.1) as

$$\boldsymbol{Z^{\star}_{VCP}} = \sum_{i=1}^{n} \upsilon_{ij}\cdot w_{ij} \leq \rho\Big\{\sum_{l=1}^{m}\pi^{l}\cdot\sum_{i=1}^{n} r^{l}_{ij}\cdot w_{ij} + \psi\sum_{i=1}^{n} g_{ij}\cdot w_{ij} + \sum_{i=1}^{n}\zeta_i\cdot w_{ij}\Big\},$$

The inequality is the result of relaxing the dual complementary slackness that implies $\upsilon_{ij} \leq \rho\big(\sum_{l=1}^{m}\pi^{l}\cdot r^{l}_{ij} + \psi\cdot g_{ij} + \zeta_i\big)$. At the same time, since $w_{ij}=1$, due to complementary slackness (i) $\sum_{i=1}^{n} r^{l}_{ij}\cdot w_{ij} = u^{l}$ and ii) $\sum_{i=1}^{n} g_{ij} = \mathrm{SLO}(S)$. Therefore,

$$\boldsymbol{Z^{\star}_{VCP}} \leq \rho\Big\{\sum_{l=1}^{m}\pi^{l}\cdot u^{l} + \psi\cdot SLO(S) + \sum_{i=1}^{n}\zeta_i\Big\}.$$

Finally, by Eq.(7.5.1), we have

$$\boldsymbol{Z}^{\star}_{\boldsymbol{VCP}} \leq \rho \cdot \boldsymbol{Z}^{RConfPD}_{\boldsymbol{VCP}} \implies \frac{\boldsymbol{Z}^{\star}_{\boldsymbol{VCP}}}{\boldsymbol{Z}^{RConfPD}_{\boldsymbol{VCP}}} \leq \rho,$$

where $\boldsymbol{Z}^{RConfPD}_{\boldsymbol{VCP}}$ is the total utilization of `RConfPD` and $\rho \geq 1$.

## 7.6 Summary

Optimal deployment of complex services in a virtualized environment is still an open problem. These services typically consist of a set of connected components, and each component may consist of multiple instances. Each instance can, in turn, be run in different virtual flavors, while the service constructed by the combination of these instances must satisfy a customer SLO.

While there have been efforts to answer the questions of when to provision additional resources in a running service, and how many resources are needed, the question of what (i.e., which combination of instances) should be provisioned has not been investigated yet. In this chapter, we offer to service providers the first system that automatically deploys component instances for complex services to maximize resource utilization at the providers' premises in the presence of customer constraints.

Our system consists of two key technologies (`RConf` and `RConfPD`), both of which build on an analytical model based on robust queueing theory to accurately model arbitrary components. With the aid of this model, `RConf` proposes an algorithm to ultimately find the *optimal* combination of component instances, such that the overall utilization of the running instances is maximized while meeting SLO.

# Chapter 8

# ARPP Resource Provisioning Evaluation

This chapter presents the evaluation results of approaches and models proposed in the previous chapter namely RConf, RConfPD and robust queueing based performance model.

First, we describe our experimental setup. Afterward, we motivate the need for profiling and present profiling results. Subsequently, we not only evaluate RConf over state-of-the-art solutions but also validate on real cloud instances. Finally, we evaluate and compare the performance and solution quality of RConfPD over state-of-the-art solutions.

## Contents

In this chapter, we evaluate RConf and RConfPD in both a real-world deployment and simulations. Specifically, we show that both solutions, constrained by a customer SLOs, do indeed find (near-)optimal configurations that increase resource utilization when compared to one-size-fits-all approaches. Further, RConfPD does find a near-optimal configuration at a computational expense order of magnitude below that of RConf.

## 8.1 Experimental Setup

The primary contribution of both RConf and RConfPD is to allow each component of a service to be run in different instance flavors if this improves the resource utilization while meeting SLOs. To obtain representative flavors for our experiments, we use a subset of the flavors offered by Amazon EC2 as listed in Table 8.1.

| Instance | vCPU | Mem (GiB) | Usecase |
|---|---|---|---|
| T2.small | 1 | 2 | General purpose |
| T2.medium | 2 | 4 | General purpose |
| T2.large | 2 | 8 | General purpose |
| M3.medium | 1 | 3.75 | Enterprise applications |
| M3.large | 2 | 7.5 | Backend servers |
| M3.xlarge | 4 | 15 | Cluster computing |
| C3.large | 2 | 3.75 | Webservers |
| C3.xlarge | 4 | 7.5 | Batch processing |
| C3.2xlarge | 8 | 15 | Distributed analytics |

Table 8.1: A summary of the instance flavors and their CPU (in cores) and RAM (in GB) resources used in our experiments.

This subset offers a variety of combinations of processors and memory, and each flavor has a different preferred use case. Generally, flavors of the M3 category are intended to be used in memory-hungry components such as the DB in our case, and C3 instances are well-suited for compute-intensive components such as the WS in our three-tiered service. We also use some general purpose instances that are not designed for a particular type of component.

To measure the real-world flavor capabilities, we deploy on Amazon EC2 the CloudSuite web serving benchmark [161] as a real-world example of a three-tier web-service as depicted in Figure 8.1. Web requests generated by httperf [162] arrive at a load balancer

(LB), for which we use HAProxy[8]. The LB forwards user requests to a web server (WS) running Elgg[9], a widely used social networking engine. Finally, the WS queries a MySQL database (DB) to fetch requested content.

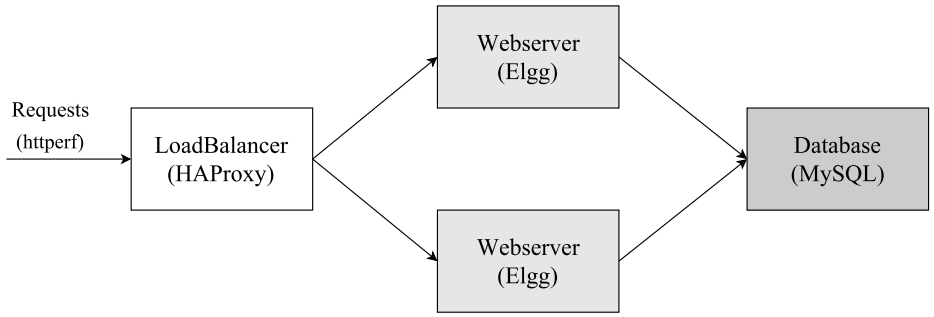



Figure 8.1: Experiment Setup

## 8.2 Profiling

Our goal in profiling instance flavors is to evaluate performance accuracy of our robust queue model. We first measure the performance in a real-world deployment and then compare our findings with the predictions obtained by a robust queueing model, as well as with the predictions obtained by employing a traditional G/G/m queue.

### 8.2.1 Methodology

We deploy our service on Amazon EC2 such that each component is running on a different instance and profile, and each component running on each flavor. To be able to profile each component of the service individually, we isolate that component, in the sense that we intentionally over-provision all other components in the service with the goal of finding the limits of the component under investigation. For example, to profile a DB on a T2.small instance, we intentionally deploy both LB and WS on C3.2xlarge instances, so that the DB is the bottleneck of the complete chain.

After that, we perform profiling by increasing the system load at a constant rate and determine the system time (i.e., the total time spent in the component between arrival and departure) of each flow for the instance at that load. Further, it is essential to calculate

[8] http://haproxy.org
[9] http://elgg.org/

the service rate of that component to determine the load of a component. The service rate computation requires deriving the processing time from the system time which is not trivial for multi-server queues since multiple servers are servicing in parallel and anyone among them can service the flow. Krivulin [155] extends the idea of *Lindley's recursion* to a G/G/m queue and derives a recursive relationship between the system, waiting and processing time. We thus implement Krivulin's equations for computing the processing time and service rate of the flows.

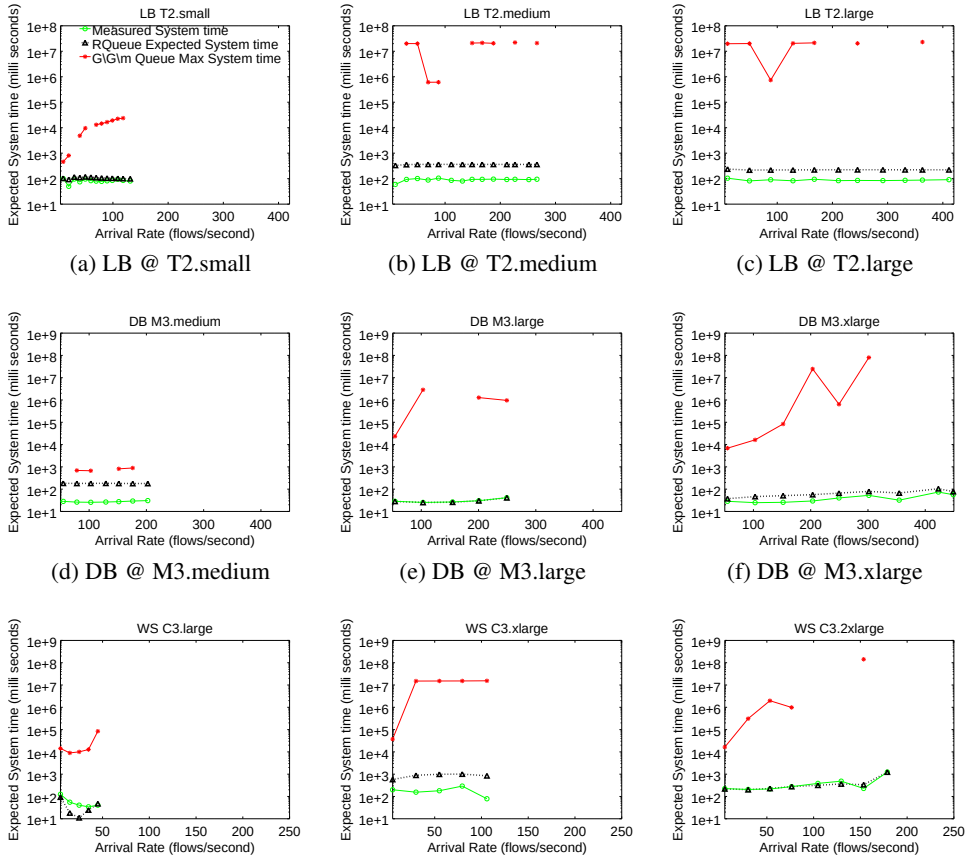
### 8.2.2 Results

Figure 8.2 shows the results of profiling EC2 instances. Each figure shows the following system times:

- The measured system time in profiling.
- The predicted system time by `RConf` robust queue model.
- The maximum system time of a G/G/m queue for all types of components we consider in our service and their most reasonable instance flavors.

Our key observations are as follows:

- **Components scale well on different flavors.** Each row in Figure 8.2 (e.g., (a) to (c)) shows one scaled type of instance for a particular component. Each step in one row roughly represents a doubling of resources, and the right-most plotted point for each measure or prediction marks the breaking point of an instance, i.e., the point at which the instance cannot handle more requests. Here, we can see that more capable instance flavors can handle higher arrival rates, which is not surprising. One interesting find is however that some instances scale very well with an increasing number of requests. For instance, we can observe that the latency for running a DB on the memory optimized instance flavors (see (d) to (f)) *decreases* with an increasing arrival rate, which may seem counter-intuitive initially. The reason for this is that the DB with increasing load also starts more worker threads internally and is thus able to handle the load appropriately. We can observe this effect up to a certain point, at which the DB begins to struggle with the load and finally reaches a breakpoint, at which it can not handle any more requests (e.g., at around 220 requests for an M3.medium instance).
- `RConf` **is accurate.** More importantly, our model can accurately predict the real-world system time of each component. `RConf` predicts the capabilities of most instances well. We see the most accurate predictions for an LB running on T2.small, a DB running on M3.large and a WS running on C3.2xlarge. In these cases, `RConf` is at most a few milliseconds off target, and in some cases (e.g., DB @ M3.large, Figure 8.2(e)) yields a perfect prediction. For the remaining six depicted flavors we




(a) LB @ T2.small (b) LB @ T2.medium (c) LB @ T2.large

(d) DB @ M3.medium (e) DB @ M3.large (f) DB @ M3.xlarge

(g) WS @ C3.large (h) WS @ C3.xlarge (i) WS @ C3.2xlarge

Figure 8.2: Measured (profiled) system time vs. `RConf` prediction vs. theoretical G/G/m system time (maximum).

also receive encouraging results, which deviate in the scale of 10*s* of milliseconds for most cases. We have profiled all three components on all nine instance flavors and this result generalizes to the combinations not shown in Figure 8.2.

- **If `RConf` is wrong, it does the right thing.** It is further worth noting that if `RConf` deviates from the actual measured values, the deviation goes in the correct direction. `RConf` is a conservative approach that *over-predicts* the time required to process a specific load with a particular combination. While over-predictions are not desirable, their downside is that `RConf` might occasionally miss a combination that can do the job with fewer resources. In contrast, if it would systematically *under-predict* the latency, it would run in the much more severe danger of violating customer SLOs by selecting instance flavors too optimistically. The only case of under-prediction is for a WS running on C3.large, and there only for a few intermediate request rates. Over-predicting in other cases balances this rare under-prediction.
- `RConf` **will adapt to dynamic load.** Our model is also able to adapt to dynamic (e.g., increasing) load. As we now know the maximum service rate for each combination (indicated by the breaking point), `RConf` will know at which point it will need to scale up a specific component. For instance, consider a service dealing with an arrival rate of 100 requests per second (rps) and has thus deployed a single M3.medium instance for the DB. If the load increases to 200 rps, `RConf` will not scale up the DB, as it knows that the M3.medium instance can handle this load well (Figure 8.2(d)). In contrast, if the load further increases to 250 rps, `RConf` will add one or more additional instance(s). Note that these instances do not necessarily have to be M3.medium again, as now `RConf` only needs to provide resources for an additional 50 rps.
- **G/G/m queues are unsuitable for our purpose.** Employing a G/G/m queue results in significantly over-predicting the system time, often by several orders of magnitude. The reason is that we can only use the maximum system time in a standard G/G/m queue, while we can not compute the average system time as done in `RConf` [76]. As a consequence, employing G/G/m queues for resource provisioning would result in a prohibitive resource over-provisioning. Concretely, deploying a DB component that matches a 30*ms* SLO can be done with one M3.large instance in `RConf`, while a G/G/m queue would deploy 1344 such instances (see Figure 8.2(e), note the log scale of the y-axis for the expected delay of a single instance predicted by a G/G/m queue). Additionally, when using a G/G/m queue, we can not find a solution always (indicated by breaks in the respective curve). The reason is that for a G/G/m queue, the maximum system time is computed as follows:

$$\tau \leq \frac{\lambda(\sigma_a^2 + \sigma_s^2/m + (1/m - 1/m^2)/\mu^2)}{2(1-\rho)} + \frac{1}{\mu} \tag{8.2.1}$$

From the above equation, we observe that $\tau$ is undefined when $\rho \geq 1$. That is, when the system is fully loaded, the system time is undefined for G/G/m.

## 8.3 RConf

In our second step, based on the results obtained from the profiling, we evaluate RConf with regards to its real-world feasibility. Here, we first let RConf find the optimal solution for the deployed service and show that RConf improves utilization over one-size-fits-all solutions. In a second step, we show based on real experiments that the solution RConf found in the first step satisfies the customer SLOs.

### 8.3.1 Methodology

In our experiment, a customer wants to set up a three-tiered web service as introduced above. Here, the customer demands that each request towards that service is handled in less than 300ms as per typical SLO in virtual infrastructures [50]. The customer also indicates that the service will experience a load of 200 rps and that she is not willing to pay for more than 20 CPU cores and 40 GB memory. To show the adaptability of RConf, we also assume that this load is not static, but increases over time, and will thus evaluate the configurations chosen by RConf for varying request rates. Note that we fix the LB to the largest instance in the experiment for all approaches, as all traffic enters our experimental deployment through that node.

We compare the solution found by RConf to several one-size-fits-all solutions. Specifically, we evaluate whether it is possible to find a solution that matches the customer SLO and budget in any of these approaches (including RConf), and how well each approach utilizes the deployed resources. More concretely, we compare RConf against configurations that only choose the largest available instances (i.e., over-provisioning), configurations that only choose the smallest available instances, and an expert configuration. In the latter, we scale up a component by adding instances that best match that component's requirements. For instance, memory-dominated instances are added to scale the DB component. Here, the expert determines a flavor that matches component's requirements in general, but it is important to note that the expert can not know the resource requirements of particular services. Thus, this allocation is a superior choice than the largest-only approach (which is not fit to any component) but is still likely to result in over-provisioning to some extent.

### 8.3.2 Results

Table 8.2 shows the solutions found by each approach for 200 rps. We can observe that only RConf and the expert choice can find solutions in this case, and further that RConf outperforms all other approaches regarding average resource utilization (ARU) of each com-

ponent. In particular, we increase utilization by 38% over over-provisioning, 50% over choosing only small instances, and 16% over the expert choice, which uses computationally powerful C3.xlarge instances for the WS and M3.large as a memory-optimized instance for the DB. Note that this utilization gain is achieved even though `RConf` over-predicts the latency for some configurations as indicated above.

| **Flavor** | **Largest** | **Smallest** | **Expert** | `RConf` |
|---|---|---|---|---|
| | | Load Balancer | | |
| C3.2XLarge | 1 | 1 | 1 | 1 |
| | | Webserver | | |
| T2.small | | 5 | | 5 |
| C3.xlarge | | | 2 | |
| C3.2xlarge | 2 | | | |
| | | Database | | |
| T2.small | | 2 | | |
| M3.medium | | | | 1 |
| M3.Large | | | 1 | |
| C3.2xlarge | 1 | | | |
| SLO (ms) | 300 | 300 | 300 | 300 |
| Latency (ms) | 299.28 | 303.93 | 293.28 | 296.3 |
| CPU/RAM used | 32/60 | 15/30 | 18/37.5 | 14/29 |
| Valid (20/40) | No (Budget) | No (SLO) | Yes | Yes |
| Valid (15/30) | No (Budget) | No (SLO) | No (Budget) | Yes |
| ARU (%) | 53.423 | 48.864 | 63.638 | 73.503 |

Table 8.2: Comparison of resource allocation and resulting utilization among different approaches under the specified budgets and latency SLO. Instance flavors not shown were not chosen by any approach.

In detail, selecting large instances results in over-provisioning with only approximately half of the resources provisioned being used. Also, the solution found in over-provisioning is violating the customer budget and thus not valid. Simultaneously, we cannot match the SLO by selecting small instances only. The reason for this is that here we are limiting the DB component to T2.small instances (which are not a good match for the DB requirements) results in increased latency for this component. As a result, no matter how many T2.small instances we deploy, we can not get below the SLO of 300ms. In contrast, the expert choice solution avoids this dilemma and selects an appropriate instance type for the DB component. We also see that with this allocation, resource utilization increases over over-provisioning

and that the solution satisfies both customer SLO and budget.

Still, `RConf` outperforms the expert choice. To find the configuration that maximizes utilization, it finds a configuration that is a mixture of smallest only (web servers) and the expert choice (database). Here, the strength of `RConf` is that it is not bound to a single instance flavor, but can instead choose from all available flavors and thus yield significant utilization gain. This utilization gain is also reflected in saving resources: `RConf` also deploys 22% fewer resources. Consequently, `RConf` can even meet stricter customer budgets. As shown in Table 8.2, unlike to `RConf`, the expert choice approach cannot find a valid solution if we decrease the budget to 15 cores and 30 GB RAM.

| Requests/second | 200 | 250 | 350 | 450 |
|---|---|---|---|---|
| Configuration DB | 1x M3.medium | 1x M3.medium + 1x T2.small | 1x M3.medium + 2x T2.small | 1x M3.medium + 3x T2.small |
| Configuration WS | 5x T2.small | 5x T2.small + 2x M3.medium | 5x T2.small + 2x M3.medium + 1x C3.xlarge | 5x T2.small + 2x M3.medium + 1x C4.xlarge + 4x T2.medium |
| Latency (ms) | 217.1 | 213.0 | 207.6 | 215.3 |

Table 8.3: `RConf` meets the pre-defined SLO requirements and scales up components as required.

Finally, we show that the configuration found by `RConf` for 200 rps is meeting the SLO set by the customer. For that, we deploy our three-tiered system under this configuration and measure the end-to-end latency of the complete service as configured by `RConf`. To demonstrate the ability to dynamically scale-up a service on demand, we increase the load towards the service sequentially. In Table 8.3, we indeed see that even for dynamic request rates, `RConf` can comply with the SLO always. Moreover, `RConf` scales up instances and runs with various combinations over time. In this case, `RConf` provides additional web servers and databases for each increase in the request rates. Here, once `RConf` receives the information about the new (increased) load, it finds the optimal configuration for the difference between the new load and the previous maximum service rate and deploys instances accordingly. For instance, the five WS running on T2.small are (almost) completely utilized with 200 rps (each having a maximum service rate of 40 rps), and thus `RConf` finds an optimal solution for the remaining 50 requests when configuring the service for the new load of 250 rps. It then deploys the found solution *in addition* to already running instances to avoid complete reconfiguration of the service. For the opposite way (decreasing load), `RConf` can similarly find the new configuration. In that case, the set of possible configurations to choose from for provisioning the new (reduced) load would be the set of combinations of the currently deployed instances. That way, the required instances will be kept running, while `RConf` can shut down unused instances. Note however that for stateful applications such as a DB or a firewall, this requires the transfer of application state to the remaining instances, which is out of the scope of this chapter.

## 8.4 RConfPD

Next, we show that RConfPD is capable of arriving at a solution much faster than RConf while providing a near-optimal configuration of a requested service.

### 8.4.1 Methodology

**Computational Complexity.** We first evaluate RConfPD in our three-tiered service against the results obtained from RConf. Then, to evaluate the scalability of both approaches, we use extended MMKP benchmarks[10] to simulate large-scale experiments [158].

There are 17 MMKP benchmarks divided into two classes, INST and RTI. Each benchmark $i$ describes $n_i$ components ($100 \leq n_i \leq 500$ for INST benchmarks, and $10 \leq n_i \leq 500$ for RTI benchmarks) and each component offers $c_i \leq 20$ configurations. Each configuration consists of up to 10 different dimensions that we assimilate as resource types, e.g., CPU cores or memory. The following table 8.4 shows the number of components and configurations (max in RTI) benchmarks.

| Benchmark | n | max $\lvert c_i \rvert$ | Total configurations | Benchmark | n | max $\lvert c_i \rvert$ | Total configurations |
|---|---|---|---|---|---|---|---|
| INST21(I21) | 100 | 10 | 1000 | RTI07(R07) | 10 | 10 | 100 |
| INST22(I22) | 100 | 10 | 1000 | RTI08(R08) | 20 | 10 | 200 |
| INST23(I23) | 100 | 10 | 1000 | RTI09(R09) | 30 | 10 | 300 |
| INST24(I24) | 100 | 10 | 1000 | RTI10(R10) | 40 | 20 | 800 |
| INST25(I25) | 100 | 20 | 2000 | RTI11(R11) | 40 | 20 | 800 |
| INST26(I26) | 100 | 20 | 2000 | RTI12(R12) | 40 | 10 | 4000 |
| INST27(I27) | 200 | 10 | 2000 | RTI13(R13) | 50 | 10 | 5000 |
| INST28(I28) | 300 | 10 | 3000 | INST29(I29) | 400 | 10 | 4000 |
| INST30(I30) | 500 | 10 | 5000 | | | | |

Table 8.4: INST and RTI benchmark information.

Hence, we have scenarios in which our solutions need to provision highly complex services with 10 to 500 components, each with up to 20 different 10-dimensional configurations. We use these benchmarks to complement the results of our three-tiered architecture. In essence, with the benchmarks, we evaluate RConf and RConfPD computational complexity on a large-scale; highly complex services that can consist of up to 500 tiers.

**Solution Quality.** Besides a lower computational complexity, RConfPD should also provide near-optimal results. To measure the solution quality, we provide two different experiments. First, we compare RConfPD and RConf performance for the benchmarks as men-

[10] http://www.es.ele.tue.nl/pareto/MMKP/benchmark_files/

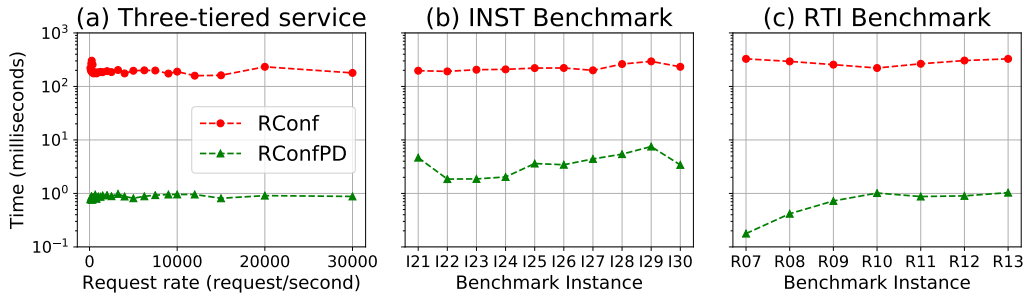



Figure 8.3: Measured running time of `RConf` vs `RConfPD`.

tioned earlier. Secondly, we emulate our initial experiment. Unfortunately, however, RTI and INST benchmarks were not thought of as a cloud service, so we modify them varying the per-resource budget to emulate the capacity of our initial three-tiered scenario. We use the configuration descriptions of each component as instance flavors and their profit as the number of requests that instance flavor can handle[11]. We then let `RConf` and `RConfPD` find (near-)optimal *combinations* of these instances. Inline with our three-tiered service evaluation for `RConf`, the aim of both solutions is maximizing resource utilization, i.e., the ratio of the number of requests that either solution provisions to the number or requests requested by the customer.

This modification allows us to vary both the customer budget as well as the request rate, similar to our `RConf` evaluation. Further, it enables a statistical comparison that goes beyond the 17 benchmarks provided by [158]. Moreover, we can now also compare `RConf` and `RConfPD` against one-size-fits-all solutions. In particular, the largest-only solution is based on the instance configuration that handles the most requests, while the smallest-only solution would pick a number of the smallest capacity instances.

### 8.4.2 Results

**Computational Complexity.** Figure 8.3 shows the measured time for various request rates and benchmarks. We find that `RConfPD` is orders of magnitude faster than `RConf` across all experiments. These results reflect our intuition behind the use of a primal-dual approximation. More importantly, for both our three-tiered service and the RTI benchmarks, `RConfPD` can determine its selected configuration within a time of one millisecond, whereas `RConf` often needs hundreds of milliseconds to arrive at its proposed configuration. Even for a

[11] Note that there is a linear relation between the profit and the resources provisioned in the benchmark, which makes this transformation feasible

highly complex service with 500 different components (INST benchmark I30), RConfPD computes a solution in at most 9*ms*. This difference makes RConfPD the go-to tool for time-sensitive applications.

**Solution Quality.** The improvement in computational complexity is only valid if at the same time RConfPD proposes a solution of high quality, close to the optimal solution of RConf. Therefore, we subsequently compare the solution quality (i.e., the ratio of the objective value achieved to the optimal objective value) of both methods. In Figure 8.4, our key observations are as follows.

First, RConfPD achieves at least 80% of the optimal objective value in all cases. Note that this does not indicate that RConfPD achieves 20% less resource utilization when compared to RConf, but is merely an indicator of how close both solutions are regarding the MMKP profit. Practically, RConfPD in many cases finds an exact solution as RConf does. Second, RConfPD is much closer to the optimal solution in the real-world scenario of our three-tiered service than in the standard MMKP benchmarks. Concretely, RConfPD achieves a solution quality of $\approx 99\%$ of the RConf solution. The reason is that here the number of components is limited to three, which leaves little room for substantial differences in choosing configurations, while the benchmarks offer much larger problem sizes. Third, note that RConfPD is unable to find a solution for benchmarks R07 and R13 due to a violation of the resource limiting constraint. The Eq. (7.2.1) formulation requires a selection from every component. In both R07 and R13, there are components with a single configuration. Hence, RConfPD has to include these configurations and then later violates the resource constraint, and does not find a feasible solution. Note, however that single configuration components are not an issue in practice, as both RConf and RConfPD are targeted at environments where there exist multiple flavors for each component.

Finally, while RConfPD often finds the same solution as RConf, it frequently chooses a different configuration with less overall resource utilization. Table 8.5 shows the selected configurations of both RConf and RConfPD for a subset of different request rates in our experiments. Here, we see that RConf chooses a different set of instances across components and request rates, which results in higher resource utilization with RConf.

| **Req./second** | 225 | 450 | 650 | 1500 | 2500 | 6250 |
|---|---|---|---|---|---|---|
| **Component** | Webserver | Webserver | Database | Database | Database | Database |
| RConf **Choice** | 3x M3.large | 7x T2.medium | 7x T2.small | 4x T2.large | 26x T2.small | 16x T2.large |
| RConfPD **Choice** | 6x T2.small | 12x T2.small | 3x C4.large | 8x M3.medium | 13x M3.medium | 32x M3.medium |
| RConf **Util.** | 99.29% | 99.21% | 94.48% | 95.99% | 97.83% | 99.99% |
| RConfPD **Util.** | 93.74% | 93.73% | 88.55% | 93.75% | 96.15% | 97.65% |

Table 8.5: Difference in configurations for RConfPD vs. RConf.

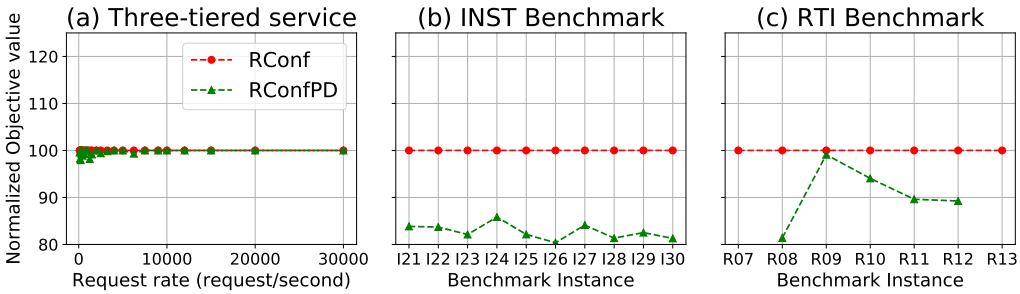



Figure 8.4: Measured solution quality of `RConf` vs. `RConfPD`.

**Solution quality in extended benchmarks.** We now evaluate our approaches with the modified benchmarks as mentioned above. We focus on two key results. First, how both `RConf` and `RConfPD` perform for a single benchmark. Second, by randomly picking 100 different customer budgets and request rates for each benchmark, we evaluate 1700 different instances for richer statistical evaluation.

Figure 8.5 shows the objective value or profit achieved by `RConf` and `RConfPD` compared with the one-fits-all configurations for the most complex benchmark, INST30 with 500 components. We can observe i) that in most cases the largest-only approach does not find a feasible configuration as it violates the customer budget; ii) `RConf` starts finding solutions with lower customer budget compared to all other approaches; iii) `RConf` always finds the best solution, while `RConfPD` and smallest-only are within 4% of that solution quality. These results are in line with what we saw when evaluating `RConf` on the three-tiered service. Note that there we also showed that the smallest-only strategy usually results in violating the SLO.

Finally, Figure 8.5 shows the statistical evaluation of the 1700 configurations in the form of a CDF, comparing the results of `RConfPD`, smallest-only and largest-only to that of `RConf`. Although no approach matches `RConf` solutions quality, `RConfPD` achieves the best results (right-most curve in the CDF). One peculiarity is that, while the largest-only strategy is not able to find any budget-satisfying solution in 81% of the cases, it beats `RConfPD` in roughly $2-3\%$ of the cases where the workload was close to a multiple of the requests the largest instance could handle.

In summary, we have initially shown that the robust queueing framework which `RConf` and `RConfPD` are built on delivers accurate profiling results for measuring flavor capabilities. Additionally, `RConf` yields an improvement of up to 50% resource utilization when compared to one-size-fits-all solutions, and simultaneously also provisions more than 20% fewer resources. For time-sensitive applications, `RConfPD` yields a speed-up factor of at

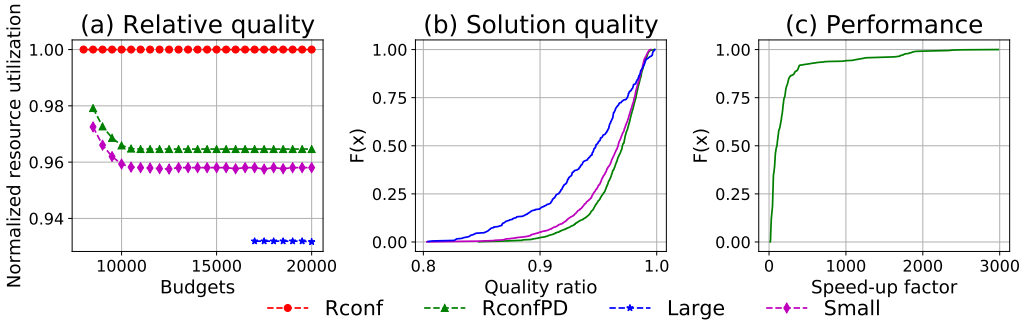



Figure 8.5: Solution quality comparison of `RConf`, `RConfPD`, large and small approaches. Furthermore, statistical evaluation of `RConfPD`, large and small approaches.

least 40× and more than 200× in ≈ 70% of the cases compared to `RConf`—and decides within at most 10*ms*. Simultaneously, it finds the same solution as `RConf` in ≈ 30% of the cases. Where it does not do so, even for huge problems with hundreds of components, it maintains at least 80% and up to 99% of the solution quality of `RConf`, while outperforming one-size-fits-all solutions.

## 8.5 Summary

In this chapter, we evaluated `RConf` and `RConfPD` along with a robust queueing based analytical model on real-world deployment as well as simulations. The profiling results clearly illustrate the adaptability and applicability of our analytical model for predicting the performance of a specific instance flavor when used for a given component in the complex service.

Our real-world experiments show that `RConf` can increase utilization by $16-50\%$ over one-size-fits-all solutions and deploys 22% fewer resources than the best of the studied approaches. Further, `RConf` manages to find valid configurations where the one-size-fits-all approaches either violate the customer SLO or the budget constraints, and thus gives the service provider additional opportunities to satisfy incoming customer requests, and thus gives the service provider additional opportunities to satisfy incoming customer requests.

For time-sensitive applications, `RConfPD` computes a near-optimal solution with up to 99% of the solution quality of `RConf` two orders of magnitude faster. Both approaches manage to find valid configurations where one-size-fits-all approaches violate either customer SLO or budget constraints. Thus, it gives the service provider additional opportunities to satisfy the incoming customer requests.

# Chapter 9

# Discussion and Future Work

This dissertation underlines the importance of optimizing both pricing and resource provisioning and introduced the challenges. As a result, previous chapters presented ARPP pricing and provisioning modules.

First, we start with the summary of whether ARPP modules addressed the challenges presented in Section 1.3. Finally, we suggest the possible enhancements to ARPP modules.

## Contents

## 9.1 Recap: Does ARPP Meet the Challenges?

In this section, we recap how ARPP addressed the challenges presented in Section 1.3. In cloud computing, customers are often *strategic*. Moreover, sealed bid auctions are susceptible to *bidder collusion*. Hence, *incentive compatible* auctions are designed to motive truthful behavior among users. RAERA satisfy not only *incentive compatibility* but also *individual rationality*. Additionally, RAERA determines allocation and prices based on the bid history and submitted bids which guarantee profit for service provider even in case of price uncertainty due to different bid distributions. As a result, RAERA does not require prior knowledge about the bid distribution of the customers.

The goal of ERM is to price resources *differentially* and at the same maximize *Nash Social Welfare* – a Pareto outcome between *efficiency* and *fairness*. In ERM, differential prices are determined based on resource demand and customer budget. Hence, the customer pays differential prices for highly demanded resources without violating their budget. Furthermore, ERM is an auction-based market and scales with a large number of customers and resource types as evident from the evaluation.

In ARPP, OFM is proposed to compute resource prices *online*, and input appears the calculation of the equilibrium prices. Similar to ERM, OFM also maximizes *Nash Social Welfare* and scales with a large number of customers and resources. Contrary to state of the art solutions, OFM neither assumes utility distribution nor the number of resources offered. Furthermore, the OFM adversarial model achieves a balance between *pessimistic* and *stochastic* models by taking *well-behaved* data observed in the Cloud. The experiments demonstrate the convergence and performance of OFM. Both ERM and OFM guarantee integer allocation and avoid the conventional approach of rounding to the nearest integer.

RConf finds an *optimal* configuration for the services using robust queueing theory-based model. The robust queueing model addressed multiple challenges in estimating performance cost of a complex service. First, the model predicts the end-to-end time for generic arrivals and departures for a complex service. Second, considers processing capacity of the configuration in performance estimation. Third, the prediction is adaptive to the load as evident from the experiments on real cloud instances. Finally, the model prediction based on arrival and departure of the traffic flows only. Therefore, the model is application independent.

The configuration found by RConf is *near optimal* and does not violate budget constraints. RConfPD is apt for edge applications which require near-instant provisioning. RConfPD finds configuration faster than RConf with a loss of optimality. The experiments on real deployment and simulation demonstrate the superiority regarding resource utilization and usage over state-of-the-art approaches. Therefore, an edge or cloud can incorporate

both RConf and RConfPD for automating the decision of finding optimal configurations for a complex service.

## 9.2 Future Work

In this dissertation, we proposed ARPP for automating resource pricing and provisioning. In this section, we present extensions to our work that we did not address in this dissertation.

### 9.2.1 ARPP Pricing

RAERA addresses price uncertainty due to different bid distributions in a sealed bid auction by constructing uncertainty set. Currently, the *central limit theorem* is the basis for an uncertainty set construction. Furthermore, RAERA can consider covariance or dependency among the resources during the uncertainty set construction. In theory, individual end users could also be buyers interested in offloading computation, (pre-)process big data, making use of IoT edge functionality. However, the scale would increase many-folds, and we consider it to be future work. Furthermore, RAERA can be extended to create an independent multi-seller marketplace.

ERM, an auction-based market for pricing resources *differentially*. The buyers' utilities are assumed to be SPLC. In cloud computing, *Leontief* utility models complementary resources. Garg [163] proves the existence of market equilibrium for buyers with *p-Leontief* – discretized Leontief utility with decreasing returns. Hence, ERM can be extended to *p-Leontief* utilities and guarantee integer allocation.

OFM is an *online* Fisher market for pricing cloud resources. OFM is based on Shymerov's alternate formulation for maximizing *Nash Social Welfare*. OFM can be extended to compute *differential prices* by extending Shymerov's formulation for multiple resource copies which not only maximize *Nash Social Welfare* but also guarantee integer allocation.

### 9.2.2 ARPP Resource Provisioning

In ARPP, RConf and RConfPD address the question *what to deploy*. Both the proposed approaches estimate performance cost of both individual components and complete service based on robust queueing theory model. Model estimation is adaptive to incoming load since uncertainty set parameters are tuned using linear regression model. However, there

are cases of over and under-prediction. Hence, recent research in regression models can be leveraged to improve the prediction accuracy of the robust queueing model.

In cloud subletting, a cloud user can rent his allocated resources temporarily for other users, and a service provider can act as the broker for the user. As a result, there is an uncertainty about resources configuration offered at the time instance i.e., resource configurations are *online*. Both `RConf` and `RConfPD` are *offline* algorithms. An *online* algorithm extension can address the challenges of cloud subletting use cases.

# Chapter 10

## Conclusion

In recent times, cloud service providers are offering complex services in addition to traditional cloud services. Moreover, these services are deployed on *edge* to improve user experience and performance. Current service pricing schemes are beneficial neither to service providers nor cloud users. Furthermore, dependency and heterogeneity among services have resulted in a complicated marketplace affecting both pricing and performance. Consequently, resource pricing and provisioning are inseparable in the context of complex services. In this dissertation, we propose `ARPP` for automatically pricing and deploying complex services in both edge and cloud environments.

This dissertation proposes `RAERA` based on robust optimization for multi-item auctions for edge computing resources. `RAERA` can guarantee the profit to providers even in the presence of *price uncertainty*. Furthermore, it satisfies *incentive compatibility* and *individual rationality* simultaneously. Therefore, `RAERA` can determine a time-dependent fair price that benefits both buyers and sellers.

This dissertation proposes `ERM`, an auction-based market with SPLC utilities for edge and cloud resource allocation. `ERM` computes *differential prices* based on the customer utility and budget. As a result, provides ability for service providers to introduce differential services. The prices are *market clearing* and maximize *Nash Social Welfare*. Furthermore, guarantee integer allocation and scales with resource types and customers.

This dissertation proposes `OFM`, an *online* Fisher marketplace for varying buyers and resources. The adversarial model incorporates *tail behavior* commonly observed in cloud computing. `OFM` achieve low regret bound and faster convergence over a time period. Furthermore, `OFM` is computationally efficient and scales with a large number of consumers and resources.

This dissertation offers `RConf` and `RConfPD` for automating the deployment of component instances of a complex service. Both approaches find configurations that maximize resource utilization without violating SLOs. The robust queueing model proposed to estimate performance cost. Furthermore, considers processing capacity of configuration and can be applied for generic arrival and departure processes unlike state of the art models in cloud computing. Both approaches manage to find valid configurations where one-size-fits-all approaches violate either the customer SLOs or the budget constraints. Thus, give the service provider additional opportunities to satisfy incoming customer requests.

## 10.1 Dissertation Impact

The author of this dissertation was the lead investigator and first author of several research papers.

In particular, the work on designing, implementing and evaluating `RConf` and `RConfPD` has been published in the following peer-reviewed international conference and journal proceedings:

- **Abhinandan S. Prasad**, David Koll, Jesus Omana Iglesias, Jordi Arjona Aroca, Volker Hilt and Xiaoming Fu, Optimal Resource Configuration of Complex Services in the Cloud. In: *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2017)*.
- **Abhinandan S. Prasad**, David Koll, Jesus Omana Iglesias, Jordi Arjona Aroca, Volker Hilt and Xiaoming Fu, RConf(PD): Automated resource configuration of complex services in the cloud. To appear: *Special Issue on Mobile, hybrid, and heterogeneous clouds for cyberinfrastructures, Future Generation Computer Systems (FGCS 2018)*.

The work `RAERA` has been published in the following peer-reviewed international conference proceedings:

- **Abhinandan S. Prasad**, Mayutan Arumaithurai, David Koll and Xiaoming Fu RAERA: A Robust Auctioning Approach for Edge Resource Allocation. In: *Proceedings of the Workshop on Mobile Edge Communications (MECOMM 2017)*.

The works `ERM` and `OFM` are **currently under submission** in the following peer-reviewed conference proceedings:

- **Abhinandan S. Prasad**, David Koll, Mayutan Arumaithurai, and Xiaoming Fu, ERM: Edge Resource Market. In: *36th International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2018 (IFIP Performance 2018)* **(under submission)**.
- **Abhinandan S. Prasad**, David Koll, Mayutan Arumaithurai, Yuming Jiang and Xiaoming Fu, OFM: Online Fisher Market. In: *36th International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2018 (IFIP Performance 2018)* **(under submission)**.

# Chapter A

# Appendix

## A.1 Optimal pricing for Fisher market

We derive optimal prices based on the duality theory. The basic concepts of duality theory can be found in the chapter 2.1.1. The convex program for Fisher market (Eq. 2.2.3)is given by

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^{n} b_i \log U_i \\ \text{s.t } U_i &= \sum_{j=1}^{m} u_{ij} x_{ij}, \forall i \in B \\ & \sum_{i=1}^{n} x_{ij} \leq 1, \forall j \in G \\ & x_{ij} \geq 0, \forall i \in B, j \in G. \end{aligned}$$

The basic idea is to derive Lagrangean dual of above equation and apply KKT [68] conditions on derived Lagrangean dual to find optimal prices. The KKT conditions can be found in chapter 2.1.2.

Let $p_j$ and $\mu_{ij}$ be the Lagrangean multipliers of constraints $\sum_{i=1}^{n} x_{ij} \leq 1$ and $x_{ij} \geq 0$ of Eq. (2.2.3) respectively. Also, $p_j \geq 0, \mu_{ij} \geq 0$. The Lagrangean dual of Eq. (2.2.3) is given by

$$\mathscr{L}(x, p, \mu) = \sum_{i=1}^{n} b_i \log U_i - \sum_{j=1}^{m} p_j \left( \sum_{i=1}^{n} x_{ij} - 1 \right) - \sum_{i=1}^{n} \sum_{j=1}^{m} \mu_{ij}(-x_{ij}) \tag{A.1.1}$$

According to KKT condition $\frac{\partial \mathscr{L}}{\partial x} = 0$, we have following:

According to KKT complementary slackness, we have

$$\begin{aligned} \forall j \in G, p_j \left( \sum_{i=1}^{n} x_{ij} - 1 \right) = 0 \\ \forall i \in B, j \in G, \mu_{ij} x_{ij} = 0. \end{aligned} \tag{A.1.2}$$

$$\begin{aligned} \frac{\partial \mathscr{L}}{\partial x} &= b_i \frac{1}{U_i} u_{ij} - p_j + \mu_{ij} \\ &= \frac{b_i u_{ij}}{U_i} - p_j + \mu_{ij} \\ \frac{\partial \mathscr{L}}{\partial x} = 0 &\implies p_j = \frac{b_i u_{ij}}{U_i} + \mu_{ij} \\ p_j &\geq \frac{b_i u_{ij}}{U_i}. \end{aligned} \tag{A.1.3}$$

In case of successful allocation, we have $x_{ij} > 0$ and substituting in Eq. (A.1.2) (complementary slackness), $\mu_{ij} = 0$. In other words, $\mu_{ij} = 0$ for successful allocation. Using $\mu_{ij} = 0$ in Eq. (A.1.3), the optimal price is given by

$$p_j = \frac{b_i u_{ij}}{U_i} \tag{A.1.4}$$

## A.2 Bregman divergence

In our case, $h(x) = x \log x - x$. Hence, $\nabla h(y) = \log y$. Bregman divergence $B_h(x,y)$ is given by

$$B_h(x,y) = h(x) - h(y) - \langle \nabla h(y), x - y \rangle$$

Substituting values, we have

$$= x \log \frac{x}{y} - x - y.$$

The Bregman divergence is minimized when $\nabla B_h(x,y) = 0$

$$\begin{aligned} \nabla B_h(x,y) &= 1 + \log x - \log y - 1 \\ &\implies x = y. \end{aligned}$$

Therefore, Bregman divergence $B_h(x,y)$ of unnormalized negative entropy function $h(x) = x \log x - x$ is minimized when $x = y$.

# Bibliography


[1] Christabel Lum. 4 Things Steve Jobs Taught Us About Cloud Computing, 2016.

[2] P M Mell and T Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

[3] Mateusz Guzek, Pascal Bouvry, and El-Ghazali Talbi. A Survey of Evolutionary Computation for Resource Management of Processing in Cloud Computing [Review Article]. *IEEE Computational Intelligence Magazine*, 10(2):53–67, May 2015.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, October 2016.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, October 2009.

[6] A. L. Jin, W. Song, P. Wang, D. Niyato, and P. Ju. Auction Mechanisms Toward Efficient Resource Sharing for Cloudlets in Mobile Cloud Computing. *IEEE Transactions on Services Computing*, 9(6):895–909, November 2016.

[7] R. Landa, M. Charalambides, R. G. Clegg, D. Griffin, and M. Rio. Self-Tuning Service Provisioning for Decentralized Cloud Applications. *IEEE Transactions on Network and Service Management*, 13(2):197–211, June 2016.

[8] Rahul Potharaju and Navendu Jain. When the Network Crumbles: An Empirical Study of Cloud Network Failures and Their Impact on Services. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, pages 15:1–15:17. ACM, October 2013.

[9] Jonatha Anselmi, Danilo Ardagna, John C. S. Lui, Adam Wierman, Yunjian Xu, and Zichao Yang. The Economics of the Cloud. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2(4):18:1–18:23, August 2017.

[10] Ranjan Pal and Pan Hui. Economic models for cloud service markets: Pricing and Capacity planning. *Theoretical Computer Science*, 496:113–124, July 2013.

[11] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient Resource Provisioning in Compute Clouds via VM Multiplexing. In *Proceedings of the 7th International Conference on Autonomic Computing*, pages 11–20. ACM, June 2010.

[12] Srinivasan Jagannathan and Kevin C. Almeroth. Price Issues in Delivering E-content On-demand. *ACM SIGecom Exchanges*, 3(2):18–27, March 2002.

[13] Zijun Zhang, Zongpeng Li, and Chuan Wu. Optimal Posted Prices for Online Cloud Resource Allocation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* , 1(1):23:1–23:26, June 2017.

[14] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. *ACM Computing Surveys*, 47(1):7:1–7:47, May 2014.

[15] Frank Ridder and Alexa Bona. Four Risky Issues When Contracting for Cloud Services, 2011.

[16] Kamal Jain, Tung Mai, and Vijay V. Vazirani. A Performance-Based Scheme for Pricing Resources in the Cloud. In *Web and Internet Economics (WINE)*, pages 281–293. Springer, December 2017.

[17] I. A. Kash and P. B. Key. Pricing the Cloud. *IEEE Internet Computing*, 20(1):36–43, January 2016.

[18] Jian Zhao, Hongxing Li, Chuan Wu, Zongpeng Li, Zhizhong Zhang, and Francis C. M. Lau. Dynamic Pricing and Profit Maximization for the Cloud with Geo-distributed Data Centers. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 118–126, April 2014.

[19] Abhinandan S. Prasad and Shrisha Rao. A Mechanism Design Approach to Resource Procurement in Cloud Computing. *IEEE Transactions on Computers*, 63(1):17–30, January 2014.

[20] Hong Xu and Baochun Li. Dynamic Cloud Pricing for Revenue Maximization. *IEEE Transactions on Cloud Computing*, 1(2):158–171, July 2013.

[21] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[22] Ishai Menache, Asuman Ozdaglar, and Nahum Shimkin. Socially Optimal Pricing of Cloud Computing Resources. In *Proceedings of the 5th International ICST Con-*

*ference on Performance Evaluation Methodologies and Tools*, pages 322–331. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), May 2011.

[23] Ashraf Al Daoud, Sachin Agarwal, and Tansu Alpcan. Brief Announcement: Cloud Computing Games: Pricing Services of Large Data Centers. In Idit Keidar, editor, *23rd International Symposium on DIStributed Computing (DISC)*, pages 309–310. Springer Berlin Heidelberg, September 2009.

[24] Ranjan Pal, Sung-Han Lin, and Leana Golubchik. The Cloudlet Bazaar Dynamic Markets for the Small Cloud. *CoRR*, abs/1704.00845, 2017.

[25] Hervé Moulin. *Fair Division and Collective Welfare* . The MIT Press, January 2003.

[26] Kamal Jain and Vijay V. Vazirani. Eisenberg–Gale markets: Algorithms and game-theoretic properties. *Games and Economic Behavior*, 70(1):84–106, September 2010.

[27] Nikhil R. Devanur, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market Equilibrium via a Primal–Dual Algorithm for a Convex Program. *Journal of the ACM*, 55(5):22:1–22:18, November 2008.

[28] James B. Orlin. Improved Algorithms for Computing Fisher's Market Clearing Prices: Computing Fisher's Market Clearing Prices. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 291–300, June 2010.

[29] Simina Brânzei, Yiling Chen, Xiaotie Deng, Aris Filos-Ratsikas, Søren Kristoffer Stiil Frederiksen, and Jie Zhang. The Fisher Market Game: Equilibrium and Welfare. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 587–593. AAAI Press, July 2014.

[30] Richard Cole and Vasilis Gkatzelis. Approximating the Nash Social Welfare with Indivisible Items. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, pages 371–380, June 2015.

[31] Richard T.B. Ma, Dah Ming Chiu, John C.S. Lui, Vishal Misra, and Dan Rubenstein. On Resource Management for Cloud Users: A Generalized Kelly Mechanism Approach. Technical report, Electrical Engineering, May 2010.

[32] Arun Anandasivam, Philipp Best, and Simon See. Customers' Preferences for Infrastructure Cloud Services. In *2010 IEEE 12th Conference on Commerce and Enterprise Computing*, pages 144–149. IEEE, November 2010.

[33] Vijay V. Vazirani and Mihalis Yannakakis. Market Equilibrium Under Separable, Piecewise-linear, Concave Utilities. *Journal of the ACM*, 58(3):10:1–10:25, June 2011.

[34] Nima Anari, Tung Mai, Shayan Oveis Gharan, and Vijay V. Vazirani. Nash Social Welfare for Indivisible Items under Separable, Piecewise-Linear Concave Utilities. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2274–2290. SIAM, January 2018.

[35] Sergei Chichin, Quoc Bao Vo, and Ryszard Kowalczyk. Towards Efficient Greedy Allocation Schemes for Double-Sided Cloud Markets. In *2015 IEEE International Conference on Services Computing*, pages 194–201. IEEE, June 2015.

[36] Sourav Chakraborty, Nikhil R. Devanur, and Chinmay Karande. Market Equilibrium with Transaction Costs. In *6th International Workshop on Internet and Network Economics (WINE)*, pages 496–504. Springer Berlin Heidelberg, December 2010.

[37] Haoming Fu, Zongpeng Li, Chuan Wu, and Xiaowen Chu. Core-Selecting Auctions for Dynamically Allocating Heterogeneous VMs in Cloud Computing. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 152–159. IEEE, June 2014.

[38] M. Bichler, J. Kalagnanam, K. Katircioglu, A. J. King, R. D. Lawrence, H. S. Lee, G. Y. Lin, and Y. Lu. Applications of Flexible Pricing in Business-to-business Electronic Commerce. *IBM Systems Journal*, 41(2):287–302, 2002.

[39] Simon Parsons, Juan A. Rodriguez-Aguilar, and Mark Klein. Auctions and Bidding: A Guide for Computer Scientists. *ACM Computing Surveys*, 43(2):10:1–10:59, January 2011.

[40] Chaithanya Bandi and Dimitris Bertsimas. Optimal Design for Multi-Item Auctions: A Robust Optimization Approach. *Mathematics of Operations Research*, 39(4):1012–1038, April 2014.

[41] Andrew V. Goldberg, Jason D. Hartline, Anna R. Karlin, Michael Saks, and Andrew Wright. Competitive auctions. *Games and Economic Behavior*, 55(2):242–269, May 2006.

[42] Adel Nadjaran Toosi, Kurt Vanmechelen, Farzad Khodadadi, and Rajkumar Buyya. An Auction Mechanism for Cloud Spot Markets. *ACM Transactions on Autonomous and Adaptive Systems*, 11(1):2:1–2:33, February 2016.

[43] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis C. M. Lau. Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs.

*IEEE/ACM Transactions on Networking*, 25(2):1034–1047, April 2017.

[44] Weijie Shi, Chuan Wu, and Zongpeng Li. RSMOA: A Revenue and Social Welfare Maximizing Online Auction for Dynamic Cloud Resource Provisioning. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, pages 41–50. IEEE, May 2014.

[45] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Francis C. M. Lau. An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing. *IEEE/ACM Transactions on Networking*, 24(4):2060–2073, August 2016.

[46] Sebastien Bubeck, Nikhil R. Devanur, Zhiyi Huang, and Rad Niazadeh. Online Auctions and Multi-scale Online Learning. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 497–514. ACM, June 2017.

[47] Spyros Angelopoulos, Atish Das Sarma, Avner Magen, and Anastasios Viglas. On-Line Algorithms for Market Equilibria. In *11th Annual International Conference Computing and Combinatorics (COCOON)*, pages 596–607. Springer Berlin Heidelberg, August 2005.

[48] Yossi Azar, Niv Buchbinder, and Kamal Jain. How to Allocate Goods in an Online Market? *Algorithmica*, 74(2):589–601, February 2016.

[49] Mohammad Hossein Bateni, Yiwei Chen, Dragos Florin Ciocan, and Vahab Mirrokni. Fair Resource Allocation in A Volatile Marketplace. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 819–819. ACM Press, July 2016.

[50] Buyya, Rajkumar and Broberg, James and Goscinski, Andrzej M. *Cloud Computing: Principles and Paradigms*, volume 87. John Wiley & Sons, 2010.

[51] Simon J. Malkowski, Markus Hedwig, Jack Li, Calton Pu, and Dirk Neumann. Automated Control for Elastic N-tier Workloads Based on Empirical Modeling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, pages 131–140. ACM Press, June 2011.

[52] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic Resource Provisioning for Cloud-based Software. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 95–104. ACM Press, June 2014.

[53] Cheng Chen, Jordi Arjona Aroca, and Diego Lugones. RobOps: Robust Control for Cloud-Based Services. In *Service-Oriented Computing*, pages 690–705. Springer

International Publishing, October 2017.

[54] Rui Han, Li Guo, Moustafa M. Ghanem, and Yike Guo. Lightweight Resource Scaling for Cloud Applications. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing ((CCGRID))*, pages 644–651. IEEE, May 2012.

[55] M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, and S. L. D. Gudreddi. Integrated and Autonomic Cloud Resource Scaling. In *2012 IEEE Network Operations and Management Symposium*, pages 1327–1334. IEEE, April 2012.

[56] Sheng Di, Derrick Kondo, and Walfredo Cirne. Google hostload prediction based on Bayesian model with optimized feature combination. *Journal of Parallel and Distributed Computing*, 74(1):1820–1832, January 2014.

[57] Daniel Jacobson, Danny Yuan, and Joshi Neeraj. Scryer: Netflixâ€™s Predictive Auto Scaling Engine. The Netflix Tech Blog. Retrieved on: April, 2016.

[58] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. Agile Dynamic Provisioning of Multi-tier Internet Applications. *ACM Transactions on Autonomous and Adaptive Systems*, 3(1):1:1–1:39, March 2008.

[59] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, Model-driven Autoscaling for Cloud Applications. In *11th International Conference on Autonomic Computing*, pages 57–64. USENIX Association, June 2014.

[60] Xiaoke Wang, Chuan Wu, Franck Le, Alex Liu, Zongpeng Li, and Francis Lau. Online VNF Scaling in Datacenters. In *2016 IEEE 9th International Conference on Cloud Computing(CLOUD)*, pages 140–147. IEEE, June 2016.

[61] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic Pricing and Traffic Engineering for Timely Inter-Datacenter Transfers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 73–86. ACM Press, August 2016.

[62] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[63] 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.

[64] Euiwoong Lee. APX-hardness of maximizing Nash social welfare with indivisible

items. *Information Processing Letters*, 122:17–20, June 2017.

[65] Dimitris Bertsimas, John N. Tsitsiklis, John Tsitsiklis, Dimitris Bertsimas, and John Tsitsiklis. *Introduction to Linear Optimization (Athena Scientific Series in Optimization and Neural Computation, 6)*. Athena Scientific, 1st edition, 1997.

[66] Gilbert Strang. *Calculus*. Wellesley College, 1991.

[67] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. Course Technology, first edition, 1996.

[68] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[69] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[70] Chaithanya Bandi, Dimitris Bertsimas, and Nataly Youssef. Robust Queueing Theory. *Operations Research*, 63(3):676–700, April 2015.

[71] A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

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

[73] Francesca Maggioni, Florian A. Potra, and Marida Bertocchi. A scenario-based framework for supply planning under uncertainty: stochastic programming versus robust optimization approaches. *Computational Management Science*, 14(1):5–44, January 2017.

[74] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[75] Kishor Shridharbhai Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications, 2nd Edition*. Wiley, 2001.

[76] R. B. Cooper. *Introduction to Queueing Theory*. North-Holland, New York, NY, second edition, 1981.

[77] Fèlix Pollaczek. *Problémes stochastiques posés par le phénoméne de formation d́une queue d́attente à un guichet et par des phénomènes apparentés*. Gauthier-Villars, 1957.

[78] Gennady Samorodnitsky and Murad S. Taqqu. *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance (Stochastic Modeling Series)*. Chapman and Hall/CRC, 1994.

[79] Valerio Di Valerio, Valeria Cardellini, and Francesco Lo Presti. Optimal Pricing and Service Provisioning Strategies in Cloud Systems: A Stackelberg Game Approach. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 115–122. IEEE, June 2013.

[80] Cheng Wang, Neda Nasiriani, George Kesidis, Bhuvan Urgaonkar, Qian Wang, Lydia Y. Chen, Aayush Gupta, and Robert Birke. Recouping Energy Costs From Cloud Tenants: Tenant Demand Response Aware Pricing Design. In *Proceedings of the 2015 ACM Sixth International Conference on Future Energy Systems*, pages 141–150. ACM, July 2015.

[81] Sijia Gu, Zongpeng Li, Chuan Wu, and Chuanhe Huang. An Efficient Auction Mechanism for Service Chains in the NFV Market. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, April 2016.

[82] G. Xilouris, E. Trouva, F. Lobillo, J. M. Soares, J. Carapinha, M. J. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro, Y. Rebahi, and A. Kourtis. T-NOVA: A Marketplace for Virtualized Network Functions. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE, June 2014.

[83] Salvatore D'Oro, Sergio Palazzo, and Giovanni Schembra. Orchestrating Softwarized Networks with a Marketplace Approach. *Procedia Computer Science*, 110:352–360, 2017.

[84] Gagan Goel and Vijay V. Vazirani. A Perfect Price Discrimination Market Model with Production, and a Rational Convex Program for It. *Mathematics of Operations Research*, 36(4):762–782, November 2011.

[85] Weijie Shi, Chuan Wu, and Zongpeng Li. An Online Mechanism for Dynamic Virtual Cluster Provisioning in Geo-distributed Clouds. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, April 2016.

[86] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis C. M. Lau. Online Stochastic Buy-Sell Mechanism for VNF Chains in the NFV Market. *IEEE Journal on Selected Areas in Communications*, 35(2):392–406, February 2017.

[87] Ruiting Zhou, Zongpeng Li, Chuan Wu, and Zhiyi Huang. An Efficient Cloud Market

Mechanism for Computing Jobs With Soft Deadlines. *IEEE/ACM Transactions on Networking*, 25(2):793–805, April 2017.

[88] Avrim Blum, Tuomas Sandholm, and Martin Zinkevich. Online Algorithms for Market Clearing. *Journal of the ACM*, 53(5):845–879, September 2006.

[89] Christina Delimitrou and Christos Kozyrakis. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 473–488. ACM, March 2016.

[90] Lei Jiao, Antonia Tulino, Jaime Llorca, Yue Jin, and Alessandra Sala. Smoothed Online Resource Allocation in Multi-tier Distributed Cloud Networks. *IEEE/ACM Transactions on Networking*, 25(4):2556–2570, August 2017.

[91] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris. Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing(CCGRID)*, pages 34–41. IEEE, May 2013.

[92] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. Dependable Horizontal Scaling Based on Probabilistic Model Checking. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID)*, pages 31–40. IEEE, May 2015.

[93] Tristan Glatard, Johan Montagnat, Diane Lingrand, and Xavier Pennec. Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR. *The International Journal of High Performance Computing Applications*, 22(3):347–360, August 2008.

[94] Tram Truong Huu and Johan Montagnat. Virtual Resources Allocation for Workflow-Based Applications Distribution on a Cloud Infrastructure. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, pages 612–617. IEEE, May 2010.

[95] Jia Yu and Rajkumar Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *ACM SIGMOD Record*, 34(3):44, September 2005.

[96] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J. Epema. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. *Future Generation Computer Systems*, 29(1):158–169, January 2013.

[97] Amandeep Verma and Sakshi Kaushal. Deadline Constraint Heuristic-based Genetic Algorithm for Workflow Scheduling in Cloud. *International Journal of Grid and Utility Computing*, 5(2):96–106, March 2014.

[98] Zhiming Zhao, Paul Martin, Junchao Wang, Ari Taal, Andrew Jones, Ian Taylor, Vlado Stankovski, Ignacio Garcia Vega, George Suciu, Alexandre Ulisses, and Cees de Laat. Developing and Operating Time Critical Applications in Clouds: The State of the Art and the SWITCH Approach. *Procedia Computer Science*, 68:17–28, 2015.

[99] Cisco. Cisco visual networking index: Forecast and methodology, 2014-2019 white paper. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html`, 2016.

[100] William Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance*, 16(1):8–37, 1961.

[101] Chaithanya Bandi and Dimitris Bertsimas. Tractable stochastic analysis in high dimensions via robust optimization. *Mathematical Programming*, 134(1):23–70, June 2012.

[102] Sushil Bikhchandani and Joseph M. Ostroy. The Package Assignment Model. *Journal of Economic Theory*, 107(2):377–406, December 2002.

[103] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube Traffic Characterization: A View from the Edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 15–28. ACM, October 2007.

[104] Chunfeng Cui, Hui Deng, Deutsche Telekom, Uwe Michel, Herbert Damker, Ivano Guardini, Elena Demaria, Roberto Minerva, and Antonio Manzalini. Network Functions Virtualisation. Technical report, ETSI, 2012.

[105] Weisong Shi and Schahram Dustdar. The Promise of Edge Computing. *Computer*, 49(5):78–81, May 2016.

[106] Mayutan Arumaithurai, Jiachen Chen, Edo Monticelli, Xiaoming Fu, and Kadangode K. Ramakrishnan. Exploiting ICN for Flexible Management of Software-Defined Networks. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 107–116. ACM, September 2014.

[107] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is It Still Possible to Extend TCP? In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, pages 181–194.

ACM Press, November 2011.

[108] Abhinandan S. Prasad, David Koll, Jesus Omana Iglesias, Jordi Arjona Aroca, Volker Hilt, and Xiaoming Fu. Optimal Resource Configuration of Complex Services in the Cloud. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 42–53. IEEE, May 2017.

[109] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16. ACM, August 2012.

[110] Neda Nasiriani, Cheng Wang, George Kesidis, Bhuvan Urgaonkar, Lydia Y. Chen, and Robert Birke. On Fair Attribution of Costs Under Peak-Based Pricing to Cloud Tenants. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2(1):3:1–3:28, November 2016.

[111] Rini T. Kaushik, Prasenjit Sarkar, and Abdullah Gharaibeh. Greening the Compute Cloud's Pricing Plans. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, pages 6:1–6:5. ACM, November 2013.

[112] Zhenhua Liu, Iris Liu, Steven Low, and Adam Wierman. Pricing Data Center Demand Response. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):111–123, June 2014.

[113] Marian Mihailescu and Yong Meng Teo. Dynamic Resource Pricing on Federated Clouds. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, pages 513–517. IEEE, May 2010.

[114] Bruno Codenotti, Benton Mccune, Sriram Pemmaraju, Rajiv Raman, and Kasturi Varadarajan. An Experimental Study of Different Approaches to Solve the Market Equilibrium Problem. *Journal of Experimental Algorithmics*, 12:3.3:1–3.3:21, August 2008.

[115] Bruno Codenotti and Kasturi Varadarajan. Efficient Computation of Equilibrium Prices for Markets with Leontief Utilities. In *31st International Colloquium Automata, Languages and Programming (ICALP)*, pages 371–382. Springer Berlin Heidelberg, July 2004.

[116] Philippe Menanteau, Dominique Finon, and Marie-Laure Lamy. Prices versus quantities: choosing policies for promoting the development of renewable energy. *Energy Policy*, 31(8):799–812, June 2003.

[117] Pablo del Río González. Ten years of renewable electricity policies in Spain: An

analysis of successive feed-in tariff reforms. *Energy Policy*, 36(8):2917–2929, August 2008.

[118] Xiaotie Deng, Christos Papadimitriou, and Shmuel Safra. On the complexity of price equilibria. *Journal of Computer and System Sciences*, 67(2):311–324, September 2003.

[119] Vijay V. Vazirani. The Notion of a Rational Convex Program, and an Algorithm for the Arrow-debreu Nash Bargaining Game. *Journal of the ACM*, 59(2):7:1–7:36, April 2012.

[120] B.Curtis Eaves. A finite algorithm for the linear exchange model. *Journal of Mathematical Economics*, 3(2):197–203, July 1976.

[121] Vilfredo Pareto. *Cours d'Economie Politique*. 1896.

[122] V. I. Shmyrev. An Algorithm for Finding Equilibrium in the Linear Exchange Model with Fixed Budgets. *Journal of Applied and Industrial Mathematics*, 3(4):505–518, December 2009.

[123] Richard Cole, Nikhil Devanur, Vasilis Gkatzelis, Kamal Jain, Tung Mai, Vijay V. Vazirani, and Sadra Yazdanbod. Convex Program Duality, Fisher Markets, and Nash Social Welfare. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 459–460. ACM, June 2017.

[124] Nikhil R. Devenur and Thomas P. Hayes. The Adwords Problem: Online Keyword Matching with Budgeted Bidders Under Random Permutations. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, pages 71–78. ACM, July 2009.

[125] Nikhil R. Devanur. Online Algorithms with Stochastic Input. *ACM SIGecom Exchanges*, 10(2):40–49, June 2011.

[126] Elad Hazan. Introduction to Online Convex Optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, August 2016.

[127] Heinz H. Bauschke, Jérôme Bolte, and Marc Teboulle. A Descent Lemma Beyond Lipschitz Gradient Continuity: First-Order Methods Revisited and Applications. *Mathematics of Operations Research*, 42(2):330–348, May 2017.

[128] Santiago R. Balseiro, Jon Feldman, Vahab Mirrokni, and S. Muthukrishnan. Yield Optimization of Display Advertising with Ad Exchange. *Management Science*, 60(12):2886–2907, October 2014.

[129] John Wilkes. More Google cluster data. Google research blog, Novem-

ber 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[130] Arash Asadpour and Amin Saberi. An Approximation Algorithm for Max-Min Fair Allocation of Indivisible Goods. *SIAM Journal on Computing*, 39(7):2970–2989, May 2010.

[131] Rupert Freeman, Seyed Majid Zahedi, and Vincent Conitzer. Fair and Efficient Social Choice in Dynamic Settings. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4580–4587. AAAI Press, August 2017.

[132] Benjamin Birnbaum, Nikhil R. Devanur, and Lin Xiao. Distributed Algorithms via Gradient Descent for Fisher Markets. In *Proceedings of the 12th ACM Conference on Electronic Commerce*, pages 127–136. ACM, June 2011.

[133] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[134] 2ndwatch.com. Top 30 Most Popular AWS Products 2017. Cloud Resources. Posted at `http://2ndwatch.com/wp-content/uploads/2018/01/2017Top30AWSProducts.pdf`.

[135] Shipra Agrawal and Nikhil R. Devanur. Fast Algorithms for Online Stochastic Convex Programming. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1405–1424. SIAM, January 2015.

[136] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic Regret Algorithms for Online Convex Optimization. *Machine Learning*, 69(2-3):169–192, December 2007.

[137] Huahua Wang and Arindam Banerjee. Online alternating direction method (longer version). *CoRR*, abs/1306.3721, 2013.

[138] Yifei Zhu, Silvery Fu, Jiangchuan Liu, and Yong Cui. Truthful Online Auction for Cloud Instance Subletting. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2466–2471. IEEE, June 2017.

[139] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control (Revised Edition)*. Holden-Day, 1976.

[140] Shai Shalev-Shwartz. Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, February 2011.

[141] Nathan Srebro, Karthik Sridharan, and Ambuj Tewari. On the Universality of Online Mirror Descent. In *Proceedings of the 24th International Conference on Neural*

*Information Processing Systems*, pages 2645–2653. Neural Information Processing Systems Foundation, December 2011.

[142] L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967.

[143] Benjamin Birnbaum, Nikhil R. Devanur, and Lin Xiao. New Convex Programs and Distributed Algorithms for Fisher Markets with Linear and Spending Constraint Utilities. Technical report, August 2010.

[144] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated Control for Elastic Storage. In *Proceedings of the 7th International Conference on Autonomic Computing*, pages 1–10. ACM, June 2010.

[145] Md Mostofa Akbar, M. Sohel Rahman, M. Kaykobad, E.G. Manning, and G.C. Shoja. Solving the Multidimensional Multiple-choice Knapsack Problem by Constructing Convex Hulls. *Computers & Operations Research*, 33(5):1259–1273, May 2006.

[146] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag Berlin Heidelberg, 2004.

[147] Kihong Park, Gitae Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of 1996 International Conference on Network Protocols*, pages 171–180. IEEE, October 1996.

[148] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the Self-similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[149] P. R. Jelenkovic, A. A. Lazar, and N. Semret. The Effect of Multiple Time Scales and Subexponentiality in MPEG Video Streams on Queueing Behavior. *IEEE Journal on Selected Areas in Communications*, 15(6):1052–1071, August 1997.

[150] Robert J. Adler, Raisa E. Feldman, and Murad S. Taqqu, editors. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*. Birkhäuser, 1998.

[151] John Nolan. *Stable Distributions*. Birkhauser, 2002.

[152] Bruce M. Hill. A Simple General Approach to Inference About the Tail of a Distribution. *The Annals of Statistics*, 3(5):1163–1174, 1975.

[153] Pavel Cízek, Wolfgang Härdlem, and Rafa Weron. Statistical Tools for Finance and

Insurance, 2016.

[154] D. V. Lindley. The Theory of Queues with a Single Server. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(2):277–289, April 1952.

[155] N.K. Krivulin. A Recursive Equations Based Representation for the G/G/m Queue. *Applied Mathematics Letters*, 7(3):73–77, May 1994.

[156] David P. Dobkin and Steven P. Reiss. The Complexity of Linear Programming. *Theoretical Computer Science*, 11(1):1–18, May 1980.

[157] A.H.G. RinnooyKan and J. Telgen. The Complexity Of Linear Programming. *Statistica Neerlandica*, 35(2):91–107, June 1981.

[158] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. A Fast and Scalable Multidimensional Multiple-choice Knapsack Heuristic. *ACM Transactions on Design Automation of Electronic Systems*, 18(4):51:1–51:32, October 2013.

[159] Martin E. Dyer and John Walker. Dominance in multi-dimensional multiple-choice knapsack problems. *Asia-Pacific Journal of Operational Research*, 15(2):159–168, 1998.

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

[161] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 37–48. ACM, March 2012.

[162] David Mosberger and Tai Jin. httperf —A Tool for Measuring Web Server Performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, December 1998.

[163] Jugal Garg. Market equilibrium under piecewise Leontief concave utilities. *Theoretical Computer Science*, 703:55–65, December 2017.