On the Economics of Infrastructure as a Service Cloud Providers: Pricing, Markets, and Profit Maximization

Adel Nadjaran Toosi

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

Department of Computing and Information Systems
The University of Melbourne

September 2014

Abstract

CLOUD computing has introduced a major shift in the IT delivery model by offering computing resources for hosting applications as a utility. This helps businesses and organizations to access advanced IT facilities offered by cloud providers without the expensive up-front investments necessary to establish their own infrastructure. In this context, significant research efforts have already been made that aim to minimize costs for cloud customers; less attention however, has been given to challenges and opportunities that cloud providers face when striving for profit maximization.

This thesis presents a set of novel market and economics-inspired policies, mechanisms, algorithms, and software designed to address the profit maximization problem of Infrastructure-as-a-Service (IaaS) cloud providers. Our solutions are proposed for two main types of providers: 1) those who rely solely on their own resources to serve customers and 2) those who participate in a cloud federation and benefit from resource sharing. We explore different tools and methods such as resource provisioning mechanisms, financial option markets, revenue management systems, and mechanism design methods to achieve the goal of profit maximization. Our evaluation of the proposed solutions demonstrates that IaaS cloud providers can increase their Return on Investment (ROI) while honoring Quality of Service (QoS) requirements associated with customer applications.

In summary, the key contributions of this thesis towards profit maximization for IaaS cloud providers are: 1) resource provisioning policies that assist a provider in a cloud federation in deciding whether to reject, outsource or terminate spot instances to handle incoming requests; 2) a financial option-based market mechanism designed for futures trading of resources in a federated cloud environment; 3) admission control algorithms embedded within a revenue management framework that supports a joint offering of usage-based, reservation-based and demand-oriented pricing models; 4) a multi-unit, online recurrent auction mechanism for selling spare capacity of a data center that is envy-free, truthful with high probability, and generates near optimal profit for the provider; and finally 5) an implementation of the proposed auction mechanism by identifying the Spot instance pricing as a Service (SipaaS) framework and its realization in OpenStack.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Adel Nadjaran Toosi, September 2014

Acknowledgements

PhD is a rewarding journey full of wonderful experiences that is not possible without support and encouragement of many people. And now since my journey is near to the end, I would like to take the opportunity to express my sincere thanks to all amazing people who helped me along the way.

First and foremost, I offer my profoundest gratitude to my supervisor, Professor Rajkumar Buyya, who awarded me the opportunity to pursue my studies in his group. I would like to thank him for continuous guidance, support, and encouragement throughout all rough and enjoyable moments of my PhD endeavor.

I would like to express my appreciation to the members of PhD committee: To Professor Christopher Andrew Leckie for his constructive comments and suggestions on my work. To Dr. Rodrigo N. Calheiros, for his generous and kind helps on developing my research skills, providing insightful comments and collaboration on my research, and proofreading the thesis.

I would like to thank all past and current members of the CLOUDS Laboratory, at the University of Melbourne. My deepest gratitude goes to Dr. Mohsen Amini Salehi without him I would have not had the courage to move to Australia and begin my PhD. I would like to thank William Voorsluys, and Deepak Poola, whose sincere friendship made my candidature life more enjoyable. To my friend, Yaser Mansouri, for his constructive comments and long discussions on improving my research and to Farzad Khodadadi for his generous helps in practical implementation of the auction framework. I would like to express my gratitude to Dr. Amir Vahid Dastjerdi for many helpful discussions and comments, and to Nikolay Grozev for proof-reading of this thesis and his comments and suggestions. I also like to thank in particular Atefeh Khosravi, Chanh Nguyen, Safiol-

lah Heidari, and Yali Zhao for their participation in the experimental study which further enhanced my thesis. My thanks to fellow members: Dr. Christian Vecchiola, Dr. Saurabh Kumar Garg, Dr. Suraj Pandey, Dr. Bahman Javadi, Dr. Anton Beloglazov, Dileban Karunamoorthy, Yoganathan Sivaram, Sareh Fotouhi, Maria Rodriguez, Chenhao Qu, Jungmin Jay Son, Bowen Zhou, and to visitors of the lab, Mehran Garmehi, Deborah Magalhaes, Tiago Justino, and Guilherme Rodrigues for their friendship and supports.

I would also like to express special thanks to my collaborators: Kurt Vanmechelen (University of Antwerp, Belgium), Professor Ruppa K. Thulasiram (University of Manitoba, Canada) and Professor Kotagiri Ramamohanarao (University of Melbourne, Australia).

I wish to acknowledge Australian Federal Government and its funding agencies, the University of Melbourne, Australian Research Council (ARC), Google, and CLOUDS laboratory for granting scholarships and travel supports enabled me to do the research in this thesis and attend international conferences.

I would like to give heartfelt thanks to my parents, my brothers, and my parents in law for their endless help, supports, and love. With special thanks to my younger brother, Ehsan, who has also helped a lot in designing algorithms and developing codes.

I would like to express my sincerest thanks to my Iranian friends in Australia: Afshin K., Ali F., Alireza K., Alireza N., Amir B., Bahareh M., Davood A., Fatemeh R., Fatima M., Hatef M., Hoda M., Khadijeh A., Naghmeh M., Ramin Z., Rozita Z., Saeed K., and Samira E., who helped us to forget the toughness of being far from home and family.

Lastly and most importantly, I thank my precious wife Mahsa for standing beside me, for her endurance, supports, and unwavering love that will always be in my heart. These few words are not enough to express my deepest appreciation for efforts she has done during past years.

Adel Nadjaran Toosi

September 2014

Contents

This page intentionally left blank.

List of Figures

Chapter 1
Introduction

THE rapid development of the Internet and the emergence of *cloud computing* have enabled a novel trend of purchasing and consuming Information Technology (IT) services. Cloud computing has brought new opportunities to the Information and Communication Technology (ICT) industry allowing businesses to outsource their IT facilities to cloud providers and avoid expensive up-front investments of establishing their own infrastructure and consequent costs of maintenance and upgrades. By means of cloud services, cloud customers can access all their required capabilities (i.e., computational resources, data, and applications) over the Internet, use what they need, and pay for what they use without being concerned with the underlying infrastructure (see Figure 1.1). As a result, customers experience the comfort of traditional utilities such as water, electricity, gas, and telephony. Advantages such as a utility model in addition to accessibility, scalability, and ease of management have created an industry-wide shift towards cloud computing solutions.

According to a forecast from International Data Corporation (IDC),[1] the worldwide spending on public cloud services is expected to surpass $107 billion in 2017. Figure 1.2 depicts IDC's forecasts on worldwide IT cloud services spending across three main cloud computing service models, namely, *Software-as-a-Service (SaaS)*, *Platform-as-a-Service (PaaS)*, and *Infrastructure-as-a-Service (IaaS)*. Among these different forms of delivering cloud services, IDC recognized the IaaS model as one of the fastest growing categories with compound annual growth rate of 27.2%. IaaS is a promising solution for enabling *on-demand* access to an *elastic* pool of configurable and *virtual* computational

[1]International Data Corporation, `http://www.idc.com/`.

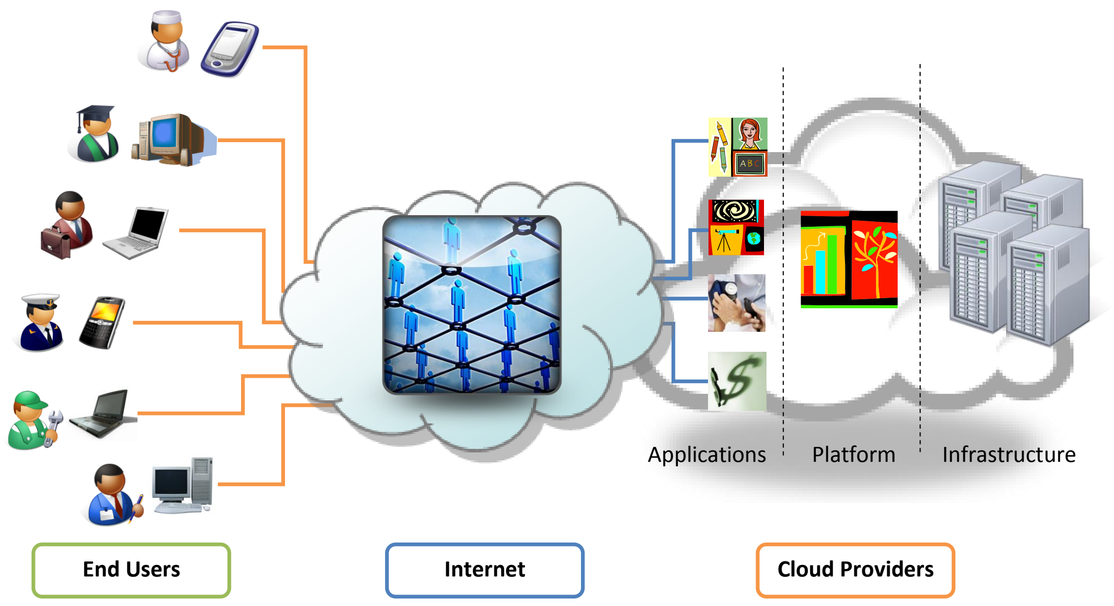


Figure 1.1: Cloud computing model.

services (e.g., computing power, storage, and networks) in a *pay-as-you-go* manner. An IaaS cloud service provider owns the data center(s) and all the required equipment and is responsible for hosting, running, and maintaining them. IaaS providers offer computational services in the form of *Virtual Machine* (VM) instances with specific resource characteristics such as computing power, memory, and disk along with operating system type and installed applications. Amazon EC2,[2] Windows Azure,[3] Rackspace,[4] Google Compute Engine,[5] and GoGrid[6] are examples of commercial IaaS offerings.

In contrast to traditional distributed systems such as Grids and Clusters, which focus on improvement of the system performance in terms of response time and throughput, cloud computing has launched a new focal point by introducing monetary and financial aspects. Thanks to the economies of scale, IaaS cloud providers can maintain large-scale data centers and offer their services at a relatively low cost. Virtualization technology also allows IaaS cloud providers to create multiple VMs on a single physical server, thus

[2] Amazon EC2, `http://aws.amazon.com/ec2/`.
[3] Windows Azure, `http://azure.microsoft.com/`.
[4] Rackspace, `http://www.rackspace.com/`.
[5] Google Compute Engine, `https://cloud.google.com/products/compute-engine/`.
[6] GoGrid,`http://www.gogrid.com/`.
[7]International Data Corporation, `http://www.idc.com/`.

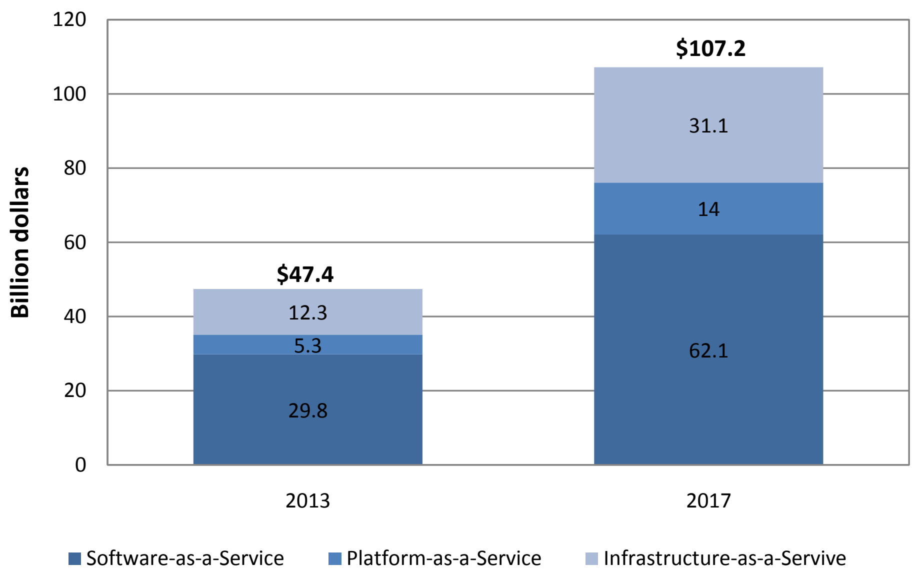


Figure 1.2: Worldwide public IT cloud services spending in billion dollars by service model.[7]

improving the resource utilization and increasing the *Return On Investment* (ROI) [8]. This however, does not remove the need for reducing cost and increasing revenue to ensure the business success and techniques to sell the services competitively while still creating profit. In other words, cloud providers need to obtain the highest possible profit from selling available capacity while they honor the Quality of Service (QoS) level agreed with the clients.

As appetite for cloud computing grows and more competitors emerge, the problem of maximizing profit becomes more complicated. Therefore, it will be increasingly important to develop market mechanisms for managing, trading, and pricing cloud resources. The profit maximization problem of IaaS cloud providers fosters an interesting stream of interdisciplinary research between computer science and economics. On the one hand, IaaS cloud providers require to minimize their cost using resource management techniques such as dynamic VM consolidation [8]. On the other hand, they must maximize revenue using techniques such as adopting differentiated pricing models [32], market segmentation [127], mechanism design [138] and demand forecasting [139].

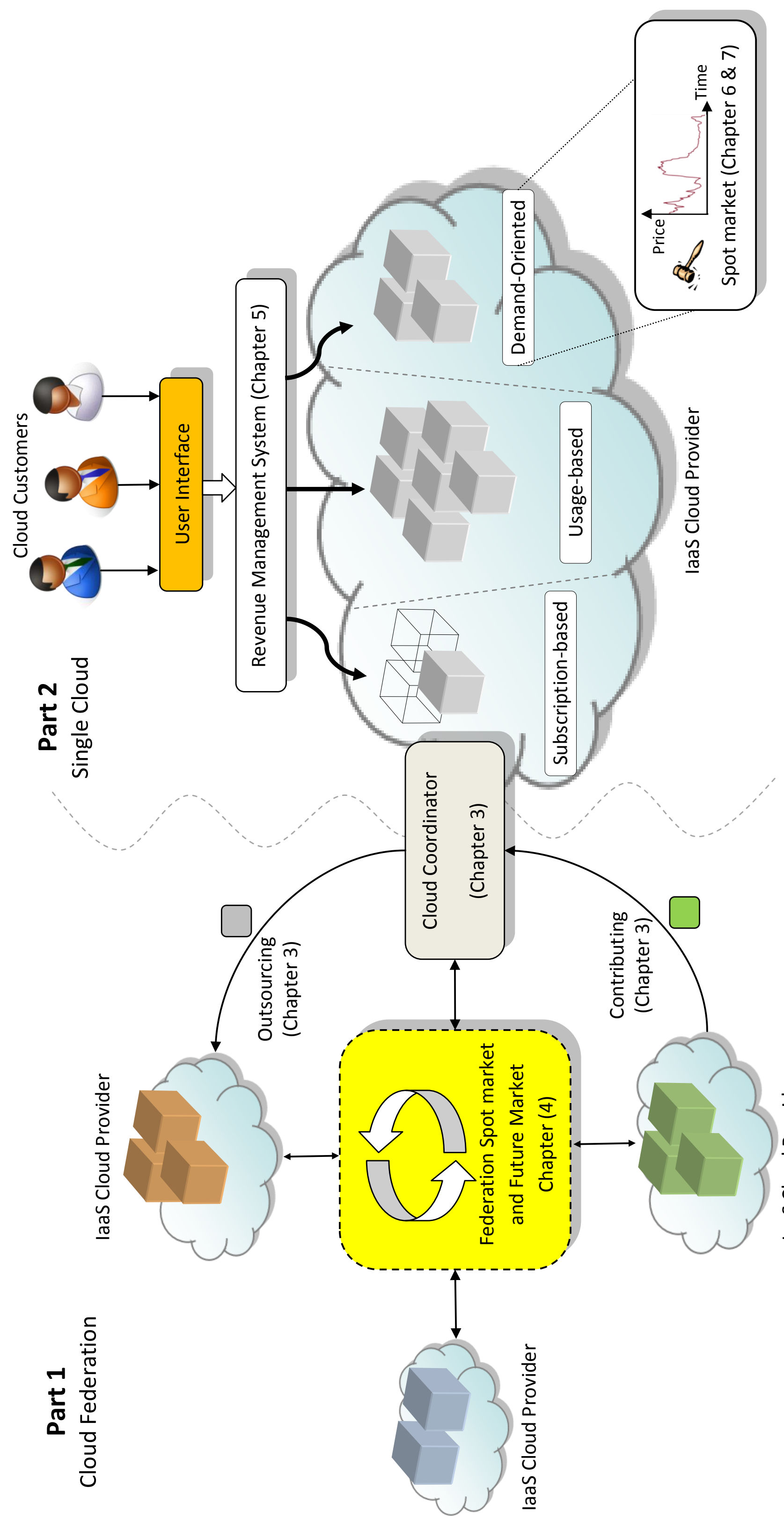


Figure 1.3: Market and economics-inspired mechanisms for IaaS cloud providers profit maximization.

This thesis focuses on the profit maximization problem of IaaS cloud providers. We investigate different market and economics-inspired mechanisms such as resource management, future markets, revenue management, and mechanism design to address the IaaS cloud providers' profit maximization problem as shown in Figure 1.3. Our proposed techniques are developed for two main different scenarios: 1) when the provider acts solely using their in-house resources to serve customers and 2) when it participates in a cloud federation and benefits from outsourcing requests. The remaining parts of this chapter detail the need for IaaS cloud provider profit maximization and discuss the research problems, contributions, and organization of the thesis.

1.1 Motivations

Cloud services, similar to many other services, are *perishable* in nature, and cannot be stored for future sale [134]. IaaS resources, such as CPU cycles, network bandwidth, and memory space, are non-storable and if not utilized at a specific point in time they are of no value and waste associated underlying data center capacity. From the perspective of the IaaS cloud provider, if a ready-to-use service is not requested and consumed during its offering period, the lost revenue cannot be reclaimed in future. The fact that computational resources sold by a cloud provider can be characterized as a *non-storable* or perishable commodity motivates cloud providers to maximize capacity utilization in order to maximize profit. Given that the IaaS cloud provider is liable to provide certain level of Quality of Service (QoS) to honor the associated Service Level Agreement (SLA), increasing utilization must be performed with a delicate and subtle consideration of the promised service guarantees.

We consider a *cloud federation* as one possible mechanism of maximizing utilization and consequently increasing profit. In order to avoid wasting non-storable compute resources, underutilized providers participating in a cloud federation can lease part of their idle capacity in the data center to other providers who require additional resources. Besides, providers are able to overcome limitations in their local infrastructure during spikes in demand by outsourcing requests to other members of the federation which results in fewer rejections of customer requests. Participating in a cloud federation raises many challenges and opportunities for the IaaS cloud providers with respect to revenue maximization, especially, when the provider uses different channels of pricing to offer the services. A part of this thesis investigates these challenges and opportunities.

The other approach for maximizing data center capacity utilization is to offer the service via multiple categories of pricing strategies. For instance, Amazon EC2 offers three pricing models to its customers: usage-based, subscription-based, and demand-oriented dynamic pricing. Using multiple pricing models with different service guarantees offers more flexible choices to customers and expands market demand. This allows for accommodating requests from customers with different types of QoS requirements and budget considerations and consequently maximizes utilization [127]. For example, offering a reservation plan is attractive for applications with steady state or predictable long term usage and customers with high-availability requirements. Services with lower QoS offered in demand-oriented dynamic pricing can attract price-sensitive users with short term, spiky, or unpredictable workloads. However, an important consequence of pricing models diversification is that it introduces a non-trivial optimization problem to the provider in relation with allocating its available data center capacity to each pricing model. Therefore, to achieve the goal of revenue maximization, providers require efficient resource management strategies.

Designing mechanisms that efficiently price perishable services is another important factor in profit maximization. The inherent perishable nature of computational resources sold by a cloud provider, combined with the fact that the demand for IaaS services is non-uniform over time, motivates the use of dynamic forms of pricing to optimize profit. Through price adjustment, based on actual and possibly forecast supply and demand conditions, customers can be incentivized to acquire idle capacity or shift demand from on-peak to off-peak hours. Consequently, both profit and consumer satisfaction can be increased. Therefore, design of a market mechanism offering a dynamic pricing model, which aims at optimizing profit by selling the spare capacity of resources available in cloud data centers is of great value.

1.2 Research challenges and Objectives

This thesis tackles research challenges arising from the following topic:

Designing market and economics-inspired algorithms and mechanisms to maximize IaaS cloud providers' profit honoring the QoS constraints associated with SLA.

As discussed in the previous section, we consider cloud federation as one possible source of profit maximization for IaaS cloud providers. A cloud federation is a scenario where cloud providers are able to establish a relationship to trade their resources in order to achieve new business advantages [100]. Therefore, part of this thesis is devoted to challenges regarding the profit maximization in federated cloud environments. Towards the IaaS cloud provider's profit maximization, the following research problems are investigated with respect to cloud federation:

- **Resource provisioning**
  - How should a provider exploit resources in a federation to dynamically increase their data center capacity?
  - Which are the most profitable resource allocation decisions when providers have different choices regarding incoming requests?
  - When and to what extent must the provider contribute local resources to the cloud federation?
- **Pricing Models**
  - When and with what price providers should sell and buy resources?
  - What are the proper contracts and pricing models in federated cloud environments?
- **Market Mechanisms**
  - What are the appropriate market mechanisms in a federated cloud environment?

  - What are the most proper market models in cloud federation which help in increasing the profit while mitigating the risks of 1) SLA violations or 2) high cost due to future price uncertainties?

To deal with the challenges associated with the above research problems, the following objectives have been delineated:

- Review, analyze, and classify the research in the area of profit maximization in federated cloud environments to gain the understanding of existing techniques and approaches.
- Study the *impact of federation* as a mechanism for maximizing a cloud provider's profit, utilization, and reputation.
- Leverage federation potentials by creating *resource provisioning policies* and efficient market mechanisms for IaaS cloud providers.
- Propose resource provisioning policies that help making decisions when providers have different choices regarding incoming requests.
- Propose a future market mechanism for the cloud federation that can be used to hedge against the risks of SLA violation and high cost of using shared resources in the cloud federation market.

The second part of this thesis focuses on the profit maximization when the IaaS cloud provider acts solely and uses their in-house resources to serve the customers. With regards to profit maximization for a single cloud provider, we consider challenges regarding two main cases: 1) market mechanism design for selling spare capacity of data centers and 2) efficient allocation of available data center capacity to both different and jointly offered pricing models. In particular, the following research problems are investigated:

- **Allocation of capacity to jointly offered pricing models**
  - How can an IaaS cloud provider maximize profit by efficiently allocating capacity to different pricing models with limited resources available in the data center?

  - How should the cloud provider control the capacity allocated to each pricing model, considering the dynamic and stochastic nature of customers demand in each trading channel?

- **Market mechanism design for selling spare capacity**
  - How should a market mechanism be designed for selling the spare capacity of the data center that maximizes profit without prior knowledge on the customers' service valuation?
  - How to design a market mechanism that measures the true value of the service while it is fair for all customers in the market?

Driven by these research problems, the following objectives have been identified:

- A comprehensive survey and classification of market and economics-inspired methods of profit maximization in IaaS cloud environments.
- Formulate the optimal capacity control problem that results in profit maximization considering the stochastic and dynamic nature of customers' demand.
- Develop a revenue management framework supporting jointly offered different pricing models that efficiently control the capacity assigned to each pricing model.
- Design a mechanism that efficiently prices perishable cloud resources in line with a provider's profit maximization goal. The mechanism must be fair (envy-free), truthful, and generate optimal profit for the provider.
- Evaluate the proposed mechanism with respect to profit generation, truthfulness, and rejection rates and quantify the efficiency loss caused by the lack of knowledge on lifetime of requests.
- Implement a framework to price and sell resources in a cloud market using the proposed pricing mechanism.

1.3 Evaluation Methodology

The evaluation of the proposed algorithms and mechanisms is carried out through two main methodologies:

1. **Discrete-event simulation**: The target entity in this thesis is an IaaS cloud provider with a data center infrastructure. Conducting repeatable large-scale experiments on a real infrastructure with real traces of customer requests is extremely difficult – if not impossible. Therefore, to ensure the repeatability and reproducibility of our experiments, as well as executing large-scale experiments, *discrete-event simulation* has been chosen as the main method for evaluation of the performance of the proposed algorithms and mechanisms. Discrete-event simulation enables us to control and conduct the experiments with different parameters to study the behavior of the mechanisms in diverse circumstances. We use and extend the CloudSim simulation toolkit [16] to support our pricing models, market mechanisms, and cloud federation to conduct our experimental simulations. In some cases, if it is possible and reasonable, our proposed algorithms are evaluated in comparison with the performance of an optimal off-line algorithm designed for the same problem.

2. **Prototyping**: Apart from simulations, it is important to evaluate the proposed mechanisms on a real infrastructure. Therefore, a prototype of the proposed auction mechanism in form of dynamic pricing framework has been implemented and used to create a market for selling unused computational resources of a cloud provider. In addition, an experimental study is conducted to evaluate the framework on a practical test environment.

Workload

Due to privacy and security reasons, IaaS cloud providers are often unaware of the type of applications running in virtualized computational resources used by cloud customers. Therefore, the workloads used for the evaluation purposes in this thesis are based on the VM requests, where the application type, executing in the VMs is unknown and we assume the IaaS cloud providers are always unaware of the life time of VMs. Moreover, there is no publicly available workload traces of real-world IaaS clouds VM requests and workload in IaaS cloud environments are often regarded as strictly confidential information as it might expose proprietary information about the provider. Therefore, to emulate the workloads in IaaS cloud data centers for the evaluation purposes, we create workloads based on models and traces suggested in the literature.

1.4 Contributions

The main contributions of this thesis can be broadly divided into 6 major categories: 1) literature review and related work analysis, 2) resource provisioning policies in federated cloud environments, 3) future market mechanism for federated cloud environments, 4) revenue management framework for the IaaS marketplaces, 5) auction mechanism design for the cloud spot market and 6) its prototyping. The **key contributions** of the thesis are as follows:

1. Resource provisioning policies in federated cloud environments:

    - Resource provisioning policies are proposed to help providers in making decisions while having different choices regarding incoming requests: rejecting, outsourcing, or terminating spot VM instances to free resources for more profitable requests. By spot VMs we mean the instances that can be terminated by providers whenever the current value for running such VMs (defined by the provider) exceeds the value that the customer is willing to pay.
    - A dynamic pricing model is proposed based on the idle capacity of the data center for selling VM resources in the cloud federation market. This dynamic pricing mechanism facilitates load balancing between federated providers, since it results in cheaper price for providers with larger amount of available resources.
    - A thorough simulation-based evaluation of the proposed policies is performed. Experimental results indicate that the proposed policies enhance the

profit, utilization, and QoS of the IaaS provider in a federated cloud environment. In addition, we evaluate the impact of spot VMs termination on the discontinuation of service usage by customers.

2. Future market mechanism for federated cloud environments:

    - A financial option-based market mechanism is proposed for trading of resources based on future contracts in a federated cloud environment. The proposed financial option-based market helps IaaS providers manage their local reservation-based market and achieve higher QoS guarantee by militating the critical and risky situation of overbooking the reserved capacity of the data center.
    - The performance of the proposed market model is evaluated in a simulation-based cloud federated environment which shows the effectiveness in increasing provider's profit without imposing any SLA violation.

3. Revenue management framework for the IaaS marketplaces:

    - The optimal capacity control problem is formulated that results in the maximization of revenue as a finite horizon Markov decision process (MDP). We propose a stochastic dynamic programming technique to compute the maximum number of reservation contracts the provider can accept from the arriving demand in order to maximize revenue. For a large capacity provider, the use of the stochastic dynamic programming technique is computationally prohibitive. We therefore present two algorithms to increase the scalability of our solution. The first increases the spatial and temporal granularity of the problem in order to solve it in a time suitable for practical online decision making. The second sacrifices accuracy to an acceptable extent through a number of simplifying assumptions on reserved capacity utilization and the lifetime of on-demand requests to increase scalability.
    - Algorithmic contributions are framed within a revenue management framework that supports jointly offering of usage-based, reservation-based and

demand-oriented pricing models. The framework incorporates an admission control system for requests of the reservation pricing model.

- We evaluate our proposed framework through large-scale simulations, driven by cluster-usage traces provided by Google. We propose a scheduling algorithm that generates VM requests based on the users' resource requirements in these traces. Under pricing conditions that are aligned with those of Amazon EC2, we demonstrate that our admission control algorithms substantially increase provider's revenue.

4. Auction mechanism for the cloud spot market:

- A multi-unit, online recurrent auction mechanism within the context of IaaS resource trading is designed. The proposed auction mechanism is envy-free, truthful with high probability, and generates near optimal profit for the provider. It adopts a greedy approach for maximizing provider profits in the online setting.
- The proposed mechanism is evaluated with respect to revenue generation, truthfulness, and bid rejection rates. Extensive simulations demonstrate that it achieves near optimality with regard to maximizing revenue without requiring prior knowledge on the order distributions. It also achieves low bid rejection rates which results in mitigating the *bidder drop problem* in online mechanisms [54]. We compare the proposed mechanism to a clairvoyant and non-clairvoyant variant of the *Optimal Single Price Auction* and to the *Uniform Price Auction*.
- A clairvoyant optimal auction mechanism based on a dynamic programming technique is designed as a benchmark to quantify the efficiency loss caused by the lack of information on the amount of time a bidder wants to hold a VM running. We present a method to dynamically compute a *reserve price*, based on a formulation of coarse grained data center power usage model that can be used by the provider within the proposed auction mechanism.

5. Spot instance pricing as a service framework:

- An implementation of the proposed auction mechanism is presented by identifying a framework called Spot instance pricing as a Service (SipaaS). SipaaS provides facilities to run a spot market using a set of web services to dynamically price VM instances.
- Add-on software solution for OpenStack platform is provided to make use of the SipaaS framework. An extension is made to Horizon – the OpenStack dashboard project – in order to add a spot market environment to OpenStack.
- An experimental study is conducted to evaluate and validate the prototype framework combined with the extension to OpenStack. The main goal of the experiment is to show that the system works flawlessly in a practical test environment.

1.5 Thesis Organization

The core chapters of this thesis are derived from a set of papers written during the PhD candidature. This thesis is organized into two main parts, the first part discusses profit maximization in federated cloud environments and the second part is dedicated to single cloud profit maximization. Figure 1.4 depicts the organization of the thesis as described below:

- Chapter 2 reviews the literature and provides the background relevant for the context of the thesis. This chapter is partially derived from:
  - **Adel Nadjaran Toosi**, Rodrigo N. Calheiros, and Rajkumar Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," ACM Computing Survey (ACM CSUR), vol. 47, no. 1, pp. 7:1–7:47, May 2014.
- Chapter 3 proposes resource provisioning policies that incorporate the outsourcing problem with option of terminating spot instances within a data center. This chapter is derived from:
  - **Adel Nadjaran Toosi**, Rodrigo N. Calheiros, Ruppa K. Thulasiram, and Rajkumar Buyya, "Resource provisioning policies to increase IaaS provider's profit

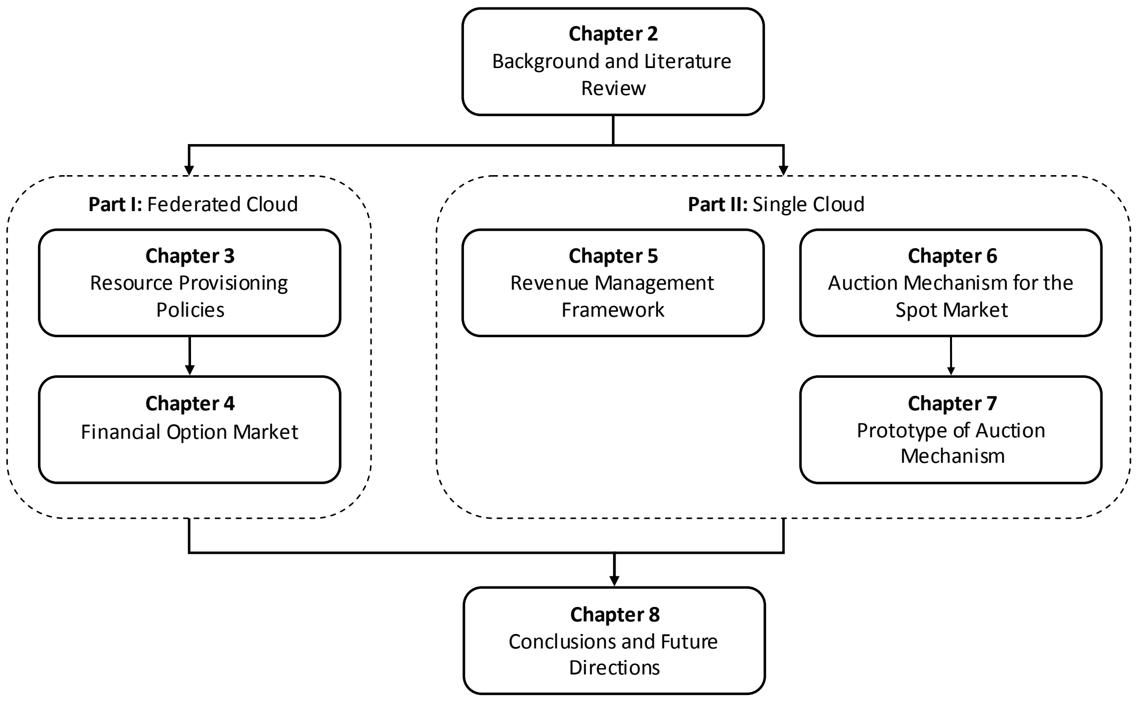

Chapter 2
Background and Literature Review
Part I: Federated Cloud
Chapter 3
Resource Provisioning Policies
Chapter 4
Financial Option Market
Part II: Single Cloud
Chapter 5
Revenue Management Framework
Chapter 6
Auction Mechanism for the Spot Market
Chapter 7
Prototype of Auction Mechanism
Chapter 8
Conclusions and Future Directions

Figure 1.4: Thesis organization.

in a federated cloud environment," in Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC'11), Banff, Canada, Sep. 2011, pp. 279–287.

- Chapter 4 proposes a financial option market model for a federated cloud environment. This chapter is derived from:
  - **Adel Nadjaran Toosi**, Ruppa K. Thulasiram, and Rajkumar Buyya, "Financial option market model for federated cloud environments," in Proceedings of the 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'12), Chicago, Illinois, USA, Nov. 2012, pp. 3–12.
- Chapter 5 presents a novel revenue management framework with optimal capacity control to maximize revenue for IaaS cloud providers. This chapter is derived from:
  - **Adel Nadjaran Toosi**, Kurt Vanmechelen, Ramamohanarao Kotagiri, and Rajkumar Buyya, "Revenue Maximization with Optimal Capacity Control in In-

frastructure as a Service Cloud Markets," IEEE Transactions on Cloud Computing (TCC), 2014, in review.

- Chapter 6 presents an auction mechanism for selling the spare capacity of the data center in a spot market. This chapter is derived from:
  - **Adel Nadjaran Toosi**, Kurt Vanmechelen, and Rajkumar Buyya, "An Auction Mechanism for a Cloud Spot Market", IEEE Transactions on Computers (TC), 2014, in review.
- Chapter 7 describes a prototype of the auction mechanism by identifying the spot instance pricing as a service framework with the associated spot market environment in OpenStack. This chapter is derived from:
  - **Adel Nadjaran Toosi**, Farzad Khodadadi, and Rajkumar Buyya, "SipaaS: Spot instance pricing as a Service Framework and its Implementation in OpenStack", Software: Practice and Experience (SPE), 2014, in preparation.
- Chapter 8 concludes the thesis with a summary of the main findings and discussion of future research directions.

Chapter 2
Background and Literature Review

The cost benefits that cloud computing offers to its customers has been discussed extensively; however, in the competitive market of cloud computing, little attention has been assigned to the challenges that cloud providers and vendors face to ensure business success. Thanks to the economies of scale, cloud providers are able to maintain large-scale data centers and to offer their services at a relatively low cost; this, however, does not eliminate the need for techniques that help providers to sell their services competitively while still creating profit. Beyond all technological advances, cloud providers endlessly require to reduce cost and increase revenue to remain in business. Among all the potential techniques to achieve such goals, we explore methods such as dynamic pricing, revenue management, resource allocation, capacity management, and cloud federation. To identify open challenges in the area and facilitate further advancements, a review of the state of the art on the aforementioned topics is presented in this chapter. We review the efforts and studies that help cloud providers to minimize cost and maximize revenue and finally conclude with a discussion on the scope of the current thesis and its positioning within the research area.

2.1 Introduction

CLOUD computing has been coined as an umbrella term to describe anything that involves delivering of computing services over the Internet. An important aim of cloud computing is to provide on-demand access to computational resources on pay-as-you-go basis similar to the way in which we obtain services from public utility services such as water, electricity, gas and telephony [15]. Such services were initially offered by commercial providers such as Amazon, Google, and Microsoft and over the years, several technologies such as Virtualization, Grid computing, and Service-Oriented Architecture (SOA) significantly contributed to make cloud computing viable [123].

Cloud computing, also known as *cloud*, refers to both the applications delivered as services over the Internet and the hardware and software in the data centers that provide those services [7]. Essentially, there are two main stakeholders in the Cloud Computing environments, which are the *Cloud providers* (service producers) and *Cloud customers* (service consumers or clients). Cloud customers can be either *software/application service providers* who have their own service consumers or *end users* (e.g., organization or businesses) who use cloud computing services directly. A cloud provider is a company or vendor that offers economically efficient cloud services using the hardware and software resources provisioned from other providers or supplied from within its own data centers. When cloud services are available to the public, it is called *public cloud* and when a cloud belongs to a business or an organization, not made available to the public, it is called *private cloud*. As shown in the Figure 2.1, there are three main types of service models usually offered by cloud providers which is also known as *cloud service stack*.

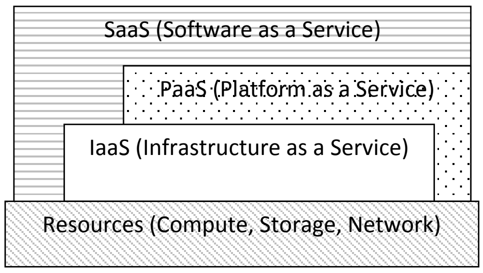

SaaS (Software as a Service)
PaaS (Platform as a Service)
IaaS (Infrastructure as a Service)
Resources (Compute, Storage, Network)

Figure 2.1: Cloud service stack.

Software as a Service (SaaS) model: As the name suggests, it provides applications and softwares to the customer in utility-based model. These applications are accessible from a thin client interface such as a Web browser. The customer does not worry about the installation, setup and running of the applications and does not manage or control the underlying cloud infrastructure or even application capabilities. A distinguished example of SaaS model is Salesforce,[1] which provides *Customer relationship management (CRM)* as a service.

Platform as a Service (PaaS) model: This model provides programming languages and tools to the customer to deploy their application onto the cloud infrastructure. The

[1]Salesforce, `www.salesforce.com`

customer does not manage or control the underlying cloud infrastructure, but has control over the deployed applications and possibly application hosting environment configurations. The example of PaaS model is Google App Engine[2] that provider facilities to build an application to be run reliably and even under heavy load in the cloud.

Infrastructure as a Service (IaaS): It provides capabilities for the customers to provision computational resources such as processing, storage, network, and other fundamental computing resources where the customer is able to deploy and run arbitrary applications. The customer does not manage or control the underlying physical infrastructure, but has control over virtualized computational hosts including operating systems, storage, and deployed applications. Examples of IaaS models are Amazon EC2,[3] Windows Azure,[4] Rackspace,[5] Google Compute Engine.[6]

The primary focus of designers in traditional distributed system such as Grids and Clusters has mostly been on the improvement of the system performance in terms of response time and throughput. The cloud computing paradigm has introduced new performance metric which is cost. This has shifted a traditional distributed system into a two-party computation, cloud providers and cloud customers, with pricing and cost as the bridge [125]. There is a large body of research devoted to minimizing cost for cloud customers using cloud computing services, however, relatively less work has been done on the provider's side to maximize revenue and reduce cost of service production. In this thesis, we study market and economics-inspired mechanisms to maximize IaaS cloud providers' revenue. Therefore, this chapter is devoted to review the related literature and to position the thesis.

The contents of this chapter are partially derived from the following published research paper:

–Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Survey* (ACM CSUR), vol. 47, no. 1, pp. 7:1–7:47, May 2014.

[2]Google App Engine, `https://appengine.google.com/`.
[3]Amazon EC2, `http://aws.amazon.com/ec2/`.
[4]Windows Azure, `http://azure.microsoft.com/`.
[5]Rackspace, `http://www.rackspace.com/`.
[6]Google Compute Engine, `https://cloud.google.com/products/compute-engine/`.

2.2 Pricing

Pricing is the process of determining the rate or fee the provider will receive in exchange for offering services or selling resources. Cloud providers can use a variety of *pricing strategies* when selling their services or resources. Finding the right pricing strategy is an important factor in running a successful business. Cloud service providers need to determine the value for their services and to capture that value through pricing. Basically, cloud computing resource pricing is a multidisciplinary area of research that embraces both economics and computer science. Studies in this area can be divided in two main groups: those that use economics and business methods to solve cloud pricing problem and those that use computer science techniques to address economic issues related to pricing. We are particularly interested in the latter group; however, we touch upon some relevant cases from the former group to attain a broader respective.

2.2.1 Pricing Factors

As shown in Figure 2.2, we recognized three main factors that cloud providers and vendors should consider when determining the price of a cloud service:

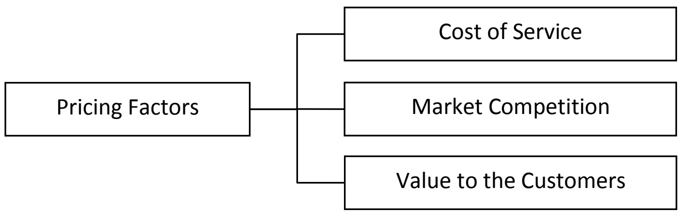


Figure 2.2: Pricing factors in cloud environments.

Cost of service: The providers must calculate the cost of service production and then add an extra percentage to set the final price of the service in a way that they achieve the targeted profit. To the best of our knowledge, there are no research studies on *cost-plus pricing* analysis in clouds and public cloud providers use their own confidential methods for service cost calculation and setting the price. However, there are few works in the literature that examine the costs of service production in cloud data centers.

Greenberg et al. [41] quantify data center costs and argue that internal data center network agility, geo-diversifying cloud provider's data centers, and market mechanisms for shaping resource consumption are the key aspects to reduce costs. Negru and Cristea [79] surveyed and analyzed existing cost models in clouds and discussed open issues related to the topic. Their guide on cost break down in today's cloud service data centers is helpful for profit maximization techniques used in this thesis.

Market competition: To remain in business, cloud providers must be aware of prices for the same services by other providers in the marketplace and set their prices competitively. Cloud computing market is moving rapidly towards a highly price-competitive environment which is termed by *perfect competition* by economists. There are few studies in the literature dealing with this problem. Pal and Hui [85] devise and analyze economic models for cloud service markets where public cloud providers jointly compete for the price and QoS levels. The competition in prices amongst the cloud providers has been envisaged by means of non-cooperative games amongst competitive cloud providers. Similarly, Roh et al. [102] study the resource pricing problem in the economic context from the perspective of cloud service providers.

Value to the Customers: To cloud customers, determining how much they are willing to pay for a service might not be related to the cost of service production by cloud providers. Setting a price for a service based on the perceived value to the customer constitutes considerable amount of subjectivity. Substantial efforts have been made by researchers of the information system sector to measure the service value of cloud computing from a customer perspective [84]. These efforts also help cloud providers to measure how well their services are leading to value and satisfaction for their customers. This is not in our scope to cover this area of research; however, we are interested in another branch of studies conducted by computer scientists to develop market mechanisms to set the price of service by asking customer to report their true valuation. Market-based pricing mechanisms such as different types of auctions that solicit truthful reports (bids) from customers and subsequently set the service price according their bids can be categorized in this area of research. In a later part of this chapter, we cover these studies when we review various dynamic pricing strategies.

2.2.2 Pricing Models

When providers understand how much it costs to provide the service, how much competitors are charging for the same service, and how customers perceive the value of services, it is time to figure out what type of *pricing model* they should utilize. The most commonly used pricing models in cloud markets, especially in infrastructure as-a-service cloud marketplaces, are *usage-based*, *subscription-based*, and *demand-oriented* pricing models (Figure 2.3).

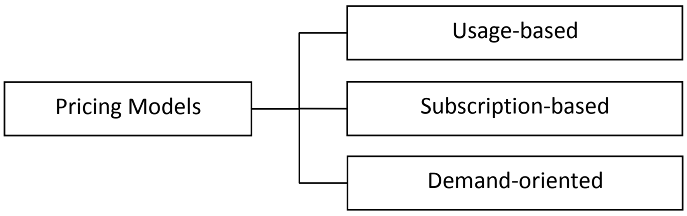


Figure 2.3: Common pricing models in IaaS cloud environments.

Usage-based pricing model: Basically, cloud computing can be defined as delivery of on-demand access to computing services on a *pay-as-yo-go* basis. This usage-based model of billing and metering of service consumption is similar to utility services such as water, electricity, gas, and telephony. A usage-based pricing model (also known as consumption-based) relies on the scheme that customers pay according to the amounts of services that they use or consume. Usage-based pricing model is the most common pricing model considered by IaaS cloud service providers. In this model, the provider quantifies the services that they provide, and charge customers accordingly. For example, an IaaS cloud provider might charge virtual machine (instance) usage per time unit, e.g., instance-minute or instance-hour or might charge storage per gigabyte per month. From the perspective of cloud customers, the pay-as-you-go pricing model offered by cloud providers is interesting in practice as it removes the upfront costs of setting up their own IT infrastructure and it allows organizations to expand or reduce their computing facilities very quickly.

In the usage-based pricing model, cloud providers often charge for services only on a fixed-rate basis. Fixed rate pricing is a relatively simple model and most often requires easily controllable cost-plus pricing strategy. There is a large body of literature on cost analysis of running applications on clouds considering the usage-based pricing model in clouds [108,117,125].

A related work by Sharma et al. [107] developed a cloud resources pricing model that uses financial option model to give a lower bound on the prices and compounded-Moores law taking into account the metrics such as initial investment, rate of depreciation, and age of resource to give a upper bound on prices for what they call cloud compute commodities.

Subscription-based pricing model: is a pricing model that allows customers to pay a subscription fee to use the service for a particular time period. This is often popular among Software-as-a-Service (SaaS) cloud providers, where vendors deliver software capability over the Internet. The idea behind subscription-based pricing is that customers pay a fee to subscribe to a service over a predefined time period and they can regularly use the service during the subscription period. Subscription-based pricing models with more or less modifications are used by IaaS cloud provider as well while it is called with different terms such as *reservation contract* or *prepaid scheme*. For example, in the case of GoGrid,[7] to use its prepaid plan, customers pay a subscription fee to reserve VM instances for monthly or annual contracts and after which the usage is free for the contract period. In Amazon Web Services,[8] the customer pays an upfront reservation fee to reserve an instance for a one or three year term and usage-based rate for that instance is heavily discounted.

Cloud providers can benefit from subscriptions because they are assured a predictable cash flow from subscribed customers for the duration of the contract. This not only provides risk-free income and removes demand uncertainty for the business, but also provides long-term usage commitment to customers. However, the provider is usually liable to provide guaranteed availability for subscriptions to honor the associated Service Level Agreement (SLA).

Niu et al. [81] propose a guaranteed cloud service model for cloud bandwidth reservation, where each customer does not require to estimate the absolute amount of bandwidth he/she needs to reserve. Their objective is to determine the optimal policy for

[7]GoGrid, `http://www.gogrid.com/`.
[8]Amazon Web Services, `http://aws.amazon.com/`.

pricing cloud bandwidth reservations in the presence of demand uncertainty such that the social welfare is maximized, that is, the sum of the expected profits for all customers and the cloud provider is maximized.

Meinl et al. [64] discuss the application of reservation systems in cloud computing environments and point out the benefits for cloud vendors as well as their customers. The authors analyzed the application of derivative pricing techniques and yield management to create a model that can be utilized in real world systems.

Mohammadi et al. [75] propose a novel reservation mechanism to protect both providers and customers from the cost overhead of over-provisioning resources. In their reservation mechanism, consumers can communicate their workload forecasts as a pre-reservation and then claim the pre-reserved resources if the need actually arises for the softly reserved resources in future. Pre-reservations capture the estimated amount of resources that will be required by a customer at a given future point of time as well as the probability of actually needing these resources. The proposed approach encompasses mechanisms to exploit the required information to be exchanged between the provider and the customer in a way that it leverages benefits of both providers and customers.

Similarly, Lu et al. [58] provide a solution for the resource reservation problem in IaaS providers with limited resource capacity. Their proposed method investigates the feasibility of each submitted reservation request and if the provider is not able to accept the request, an alternative way of accommodating the request with backward or forward shifting in time is suggested. They utilize *computational geometry* to tackle the problem.

Wang et al. [130] study the resource reservation management issues inside cloud environments. They propose an adaptive resource reservation approach by selectively accepting reservation requests. The decision is made to maximize the cloud provider revenue while it ensures the quality of service (QoS) for transactional applications.

Demand-oriented pricing model: Demand-oriented pricing model is the process of establishing a price for a service based on the level of demand. The service price is changed according to its demand in a way that when the demand is high the price goes up and when it is low the price goes down. Among all pricing models discussed here, this is the least common pricing model at real-world IaaS cloud marketplaces; however it has received the highest attention from researchers in academia due to its complexities. In the demand-oriented pricing model, the price for a service must be set based on real-time and dynamic level of demand. When done successfully, such a *dynamic pricing* model maximizes the revenue for the cloud provider.

Amazon is one of the IaaS cloud providers that publicly offers a demand-oriented pricing model for selling IaaS resources. The resources are called *spot instances* and are sold according to a dynamic pricing model that varies the price of instances in real-time based on supply and demand according to Amazon's claim.

A relevant study has done by Niyato et al. [82] where they present an economic analysis of the resource market in cloud computing environment. Three types of resource market between private customers and service providers have been considered, i.e., monopoly, competitive, and cooperative oligopoly. Repeated game model has been used to analyze the cooperation behavior of the cloud providers to reach efficient and fair profit.

Kantere et al. [49] presented an optimal pricing method achieved through the dynamic pricing for a caching service offered by the cloud service provider. They propose a novel price-demand model designed for a cloud cache and a dynamic pricing scheme for queries executed in the cloud cache. They also discuss qualitative aspects of the solution that allows the consideration of customer satisfaction together with cloud provider profit maximization.

A dynamic pricing scheme suitable for federated cloud environments with rational and self-interested parties where resource demand and supply fluctuate as customers join and leave the system has been proposed by Mihailescu and Teo [71]. They compare the performance of their proposed strategy-proof dynamic pricing model with fixed-rate usage-based pricing using simulations, and show that customer welfare and the percentage of successful requests are increased when their dynamic pricing model is applied.

As the importance of dynamic pricing in cloud pricing has been recognized by the literature [71, 125, 134], we devote the next section to discussing and reviewing dynamic pricing related work.

2.3 Dynamic Pricing

Dynamic pricing is a *time-based* and *price discrimination* scheme that allows the service provider to vary the price in real-time in response to various factors such as market demands, the time of service offering, and the type of customer. Through price adjustment based on actual (and possibly forecasted) supply and demand conditions, customers can be incentivized to acquire resource or shift demand from on-peak to off-peak hours. Consequently, from the provider point of view, both revenue and consumer satisfaction can be increased.

Additionally, dynamic pricing helps cloud providers in more-effective resource management and capacity planning. By ensuring that prices match the market conditions, fully flexible dynamic pricing mechanisms also empowers customers to manage their cost efficiently. However, this dynamism of prices makes providers' pricing decisions and customers' budget planning further challenging. This can be the main reason that currently fewer number of cloud providers are offering dynamic pricing models. Nonetheless, as competition among providers in cloud computing marketplace grows and more complex pricing models appear, dynamic pricing models would gain more popularity and approval gradually.

In general, dynamic pricing can be determined by a provider *revenue maximization* problem in a monopoly market, or by a *social welfare* maximization problem in a competitive market with multiple providers [133]. In this thesis, we study the former where we present dynamic pricing model and revenue management techniques for IaaS clouds. Social welfare can be specified as the summation of the welfare of all the individuals in the system where welfare can be measured either in terms of utility values or money, or even *Pareto efficiency*.[9] Dynamic pricing can be used to maximize welfare (the difference between the user utility and payment) of all participants in the market, e.g., customers and providers.

Mihailescu and Teo [71] present dynamic pricing scheme for a federated cloud environment aiming at maximizing individual participants' welfare in the IaaS cloud market

[9]Pareto efficiency is a state of allocation of resources in which it is not possible to give one individual more utility without giving at least one other less utility

with a large number of providers and customers. Similarly, Hassan et al. [44] study the design of theoretical game models using a price-based resource allocation strategy among the IaaS cloud providers in a federated cloud environment. They develop cooperative and non-cooperative games and examine the social welfare maximization under each game model. Menache et al. [66] use pricing as a means to maximize the social welfare in cloud environments. They show that the socially optimal operating point is unique, and can be induced by a per-unit pricing model with the same price to all customers.

One of the main techniques used in the literature to maximize cloud providers' revenue is dynamic pricing. Xu and Li [134] present an infinite horizon stochastic dynamic program to maximize the cloud provider's revenue with stochastic demand arrivals and departures. They rely on dynamic pricing as the main technique to maximize revenue. The problem of revenue maximization with dynamically allocating resources to spot markets has been investigated by Zhang et al. [139]. Supply adjustment and dynamic pricing are used as a means to maximize revenue and meet customer demand. Wang et al. [127] propose an optimal mechanism for a dynamic pricing in spot market based on the uniform price auction. The problem of optimal capacity segmentation between requests form usage-based market and dynamic pricing market has been also formulated and addressed. In order to maximize the cloud provider's revenue, a dynamic pricing model based on *genetic algorithms* has been proposed by Macías and Guitart [61]. The proposed method has been compared with demand-oriented pricing model proposed in previous works from the same authors.

As shown in Figure 2.4, there are various types of dynamic pricing strategies suggested by the literature for setting the price of cloud resources. These dynamic pricing strategies can be categorized into two main groups: *price-discovery* and *price-posted* models. In the former, the provider sets the price based on the communication with the customers, e.g., asking them to report their bid. *Auction-based* and *negotiation-based* techniques fall into this group. The latter, the price-posted model, does not necessarily require communication with the customers and the provider posts the pre-determined price which dynamically varies during the time based on some external factors such as demand or time of use. The *demand-oriented* pricing model and *yield management* techniques are categorized in the second group. In the following subsections, we discuss about various type of dynamic pricing suitable for cloud computing in more detail.

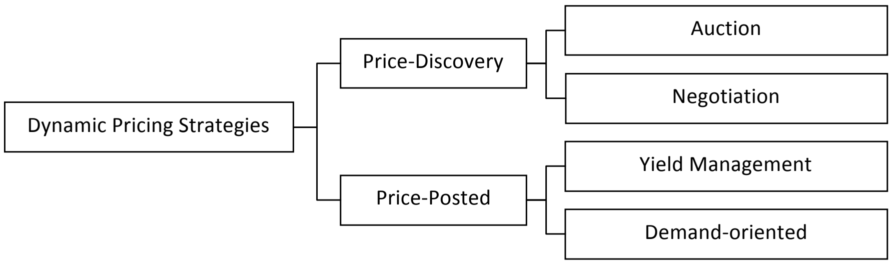


Figure 2.4: Dynamic pricing strategies in clouds.

2.3.1 Auction-based

Over the recent years, there has been a massive growth in the research of designing auctions, largely motivated by the development of the Internet. Auction is a common market mechanism with a set of rules determining prices and resource allocations on basis of bids submitted from the market participants. Auctions, and specifically *online auctions*, can be used for dynamic, value-based pricing in cloud marketplaces. Well-designed auction mechanisms are desired by the service providers since they: 1) incentivize users to reveal the service value (i.e., report the price they are willing to pay for resources), 2) ensure resources are allocated to those who value them the most, and 3) correctly price resources in line with supply and demand conditions.

Auctions can be in assorted shapes and have different characteristics such as: single-dimensional (e.g., only bid price) or multi-dimensional (e.g., bid price plus quantity), single-sided (e.g., only customers submit bids) or double-sided (e.g., double auction[10]), open-cry or sealed-bid, single-unit (e.g., a single good or service) or multi-unit (e.g., multiple units of the goods), single item (e.g., one type of service) or multi-item (e.g., combinatorial auction[11]). These have been extensively discussed and analyzed in the economics literature. Interested readers are referred to [87] for a general survey on auction mechanisms and biding from a computer science perspective.

[10]Type of auction that brings together multiple buyers and sellers where both side simultaneously submit their bids to an auctioneer.

[11]Type of auction in which participants can place bids on combinations of items rather than individual items, e.g., $5 for 3 apples and 2 oranges.

Apart from all the different types of auction that can be devised, auction designer might have specific goals in designing auction for example *truthfulness*, *revenue maximization*, *allocative efficiency*, and *fairness*. Mechanism design is a subfield of economic theory deals with devising auction mechanisms to satisfy aforementioned goals in a strategic settings – assuming that the different participants in the auction each acts *rationally* in a game theoretic sense. Here, we review some of the most common goals in designing auctions as illustrated in Figure 2.5.

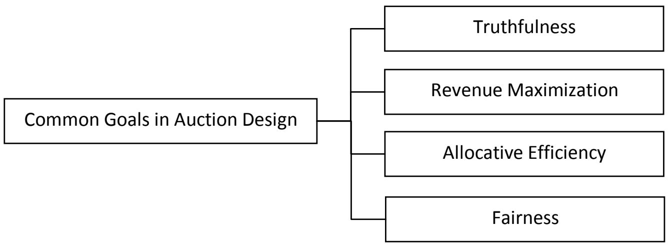


Figure 2.5: Common goals in designing auctions.

Truthfulness: An auction mechanism is *truthful* – also called *strategy-proof* or *incentive-compatible* – if the dominant bidding strategy for every bidder is to always submit its true valuation irrespective of the behavior of the other bidders. In game theoretic sense, one strategy for a given player is dominant if, regardless of what the other players do, that strategy causes highest pay-offs for that player than any other strategy.

Revenue maximization: A fundamental objective in mechanism design, and the focus of this thesis, is revenue maximization in which the mechanism designer's goal is to maximize the revenue (or some times profit) for the seller (provider). This topic in economics is referred to *optimal* mechanism design.

Optimal mechanism design can be categorized in two main groups [80]: *Bayesian optimal mechanism design* and *prior free optimal mechanism*. In Bayesian optimal mechanism design, it is assumed that the valuations of the participants in the auction are drawn from a known prior distribution. Studies in this area often formed based on the seminal work by Myerson [77]. When determining the prior distribution is not practical, convenient or even possible in advance, prior free optimal mechanism design is the case.

Allocative efficiency: In some type of mechanism design the main objective is to just allocate resources among buyers in an efficient manner, i.e., social welfare is maximized. It is well known that efficiency is optimized by Vickerey Auctions [122]; even though efficient auction mechanisms similar to Vickerey auction are theoretically sound, they are not common in practice as it does not necessarily maximize sellers' or providers' revenue. Auction mechanism often has one of the objectives of revenue maximization or allocation efficiency [80].

Fairness: There have been many different notions of *fairness* associated with the mechanism design. The main goal is to understand what is the fair way to divide the total cost (from customers' point of view) or revenue (from providers' point of view) between auction participants. In fact, fairness tries to enforce and sustain cooperation among agents willing to cooperate and to fairly divide benefits or costs of joint effort among participants. One relevant notion of fairness is *envy-freeness*. In an envy-free auction no bidder can increase its utility by adopting another bidder's outcome, that is, no bidder would be happier with someone else's outcome [38].

In practice, the use of an auction-like mechanism to sell spare capacity in IaaS cloud data centers was pioneered in late 2009 by Amazon which is called *spot market* in the literature. In Amazon's spot market, customers bid the maximum hourly price they are willing to pay to obtain a VM instance. All instances incur a uniform charge, the spot market price. According to Amazon, this price is set dynamically based on the relationship of supply and demand over time. However, Amazon has revealed little information on the pricing and allocation rules of their pricing mechanism.

In theory and research, Danak and Manno [25] present a uniform-price auction for resource allocation that suits the dynamic nature of grid systems. Mihailescu and Teo [69] investigate Amazon EC2's spot market as a case in a federated cloud environment. They argue that spot pricing used by Amazon is truthful only in a market with a single provider, and show that rational users can increase their utility by being untruthful in a federated cloud environment. Wang et al. [127] proposed an optimal recurrent auction based on the Bayesian mechanism design. The mechanism was designed in the context of optimally segmenting the provider's data center capacity between on-demand and spot market requests. A truthful dynamic auction that periodically computes the number of instances to be auctioned off in order to maximize the provider's revenue is also proposed by the same authors [128]. Unlike Amazon spot market, their approach offers guaranteed services (i.e., instances are never terminated by the provider) and constant price over time (i.e. as the price is set for the user, it remains constant as long as the user holds the instance). Their auction charges each user different price and does not generate a market-wide single price, that is, their mechanism is not fair in essence.

Prasad and Rao [91] present a cloud resource procurement approach and dynamic pricing model that automates the selection of an appropriate cloud vendor. They have presented three mechanisms to deliver allocative efficiency, incentive compatibly and individual rationality. Zaman et al. [137] have investigated the applicability of combinatorial auction mechanisms for allocation and pricing of VM instances in cloud computing. Samimi et al. [105] propose an incentive-compatible combinatorial double auction for resource allocation in cloud environments. Another work considers combinatorial auction model and is focused mainly on maximizing the seller's profit and resource trading price has presented by Wang et al. [131].

2.3.2 Negotiation-based

Even though negotiation (also known as *bargaining*) is a well-established topic of research, there is little or no automated negotiation support for cloud computing environment. Negotiation is a form of decision-making between two or more parties intended to reach an agreement in which no party can make decision independently, and therefore must make concessions to achieve a compromise [110]. Negotiation can have many attributes to negotiate over, e.g., *Quality of Service (QoS)*, but for the purpose of this section only price will be taken into account.

The main difference between negotiation and auction is that auction is an explicit set of allocation and pricing rules on the basis of bids solicited from the market participants to determine the value of objects while negotiation is about cooperating to create value of the objects [110]. Negotiation is usually done in the presence of incomplete information where parties generate offers and counteroffers during the negotiation to maximize their own benefit. Through negotiation, resource providers are given the opportunity to maximize the revenue and customers to minimize the cost. Negotiation techniques define how a party must react to offers from other parties or how offers and counteroffers must be generated during the negotiation with the ultimate goal of reaching an agreement depending on the requirements and objectives.

There is large body of literature on negotiation and there are various types of negotiation techniques and strategies. Basic concepts in automated negotiation include *negotiation agent*, *negotiation object*, *negotiation protocol*, and *negotiation strategy* [29]. The importance of each concept varies according to the negotiation context, for example, in some cases the main concern is negotiation protocol while in other cases negotiation strategy becomes important. This is not our scope to cover all aspects of negotiation; therefore we briefly review related work on automated negotiation techniques developed for cloud computing environments.

Most of related work in the area focus on establishing *Service Level Agreement (SLA)* through negotiation. An SLA clearly states the defined expectations with which the provider and the customer will do business together and price of service delivery might be one of these expectations. An example is the research conducted by Yoo and Sim [136] that propose agent-based multilateral price negotiation model for cloud service market. The proposed model includes a many-to-many negotiation protocol, and price-determining factor from service level feature. Market-driven agent and adaptive concession making strategies are proposed for negotiation where the former has full information about competitors and the later has incomplete information.

An et al. [3] present a negotiation mechanism in which agents make contracts to bind cloud resources from a provider to a customer for a fixed term interval. Each agent is able to decommit from a contract by paying a penalty to the other contract party when it finds it beneficial. Their model maximizes the social and improves the resource allocation efficiency.

Zheng et al. [141] review the state of the art and outlines a research roadmap on cloud service negotiation. They formulate three research problems, i.e., QoS measurement, QoS negotiation, and QoS enforcement, for cloud services. Their study shows when one has no knowledge of which strategy the opponent will play, a *mixed strategy* can achieve a higher utility than the *concession strategy*.

Son et al. [111] propose a multi-issue negotiation mechanism to reach an agreement on the price of a service and when to use the service. In their mechanism a negotiation agent can concurrently make multiple proposals in each negotiation round. They also characterize a utility function that models the preferences of the customer for different time slots. A case study demonstrating the benefits of using their mechanism with three pricing models of Amazon EC2 has been carried out.

2.3.3 Yield Management

Considering the *perishable* and *non-storable* nature of services in clouds, providers can benefit from maximizing resource utilization in order to maximize revenue. Services are in general perishable products because if a ready to use service is not requested and consumed during its offering period, from the perspective of the service provider, this is a lost business opportunity and the lost revenue cannot be reclaimed in future. This is entirely different from storable goods that provider can use *inventory-management* approaches to store unsold products for future selling.

Yield management (also known as *Revenue Management*) is the process of maximizing revenue from a fixed, perishable resource capacity using *pricing*, and *capacity control* techniques. During the last few decades, revenue management has witnessed significant scientific and practical advances especially in the airline and hotel industries. The literature is vast on this topic and our aim in this section is limited to review the relevant existing applications of this field to cloud computing. Interested readers can find a detailed overview of revenue management in [132].

Compared to auction mechanisms and bidding, revenue management techniques for pricing are different, here the price is determined in the marketplace without any interaction or communication between providers and customers, and it usually happens based on stochastic analysis of the demand. Therefore, we classified the revenue management techniques in price-posted model. In general, Revenue Management is mainly concerned with how best to price or allocate capacity to each market segment. Here, we review these two main techniques used for revenue management in cloud computing.

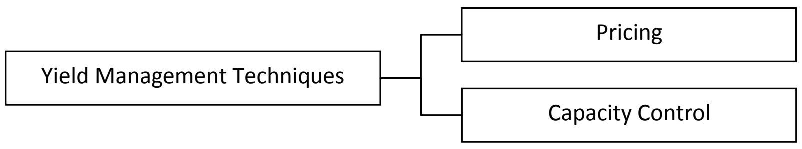


Figure 2.6: Main techniques used for revenue management in cloud environments.

Pricing: This category of revenue management involves anticipating the value created for customers and setting the price in a way that capture that value. This requires developing disciplined pricing tactics to dynamically react to changes and continually capture value and generated revenue.

Capacity control: This category of revenue management involves optimally allocating resources or capacity to each market segment in order to maximize revenue. This requires methods of forecasting demand, measuring price sensitivity of customers and possibly overbooking or admission control techniques.

Revenue management techniques have been previously used in the Grid computing [116] context. One of the early attempts to incorporate revenue management into cloud computing was made by Püschel and Neumann [92, 93]. They investigate the use of a policy-based admission control model to resource management components using techniques such as client classification and dynamic pricing. Similar work has been done by Meinl et al. [64] who applies derivative markets and yield management techniques for revenue maximization. Macías et al. [60] propose several techniques such as dynamic pricing, over-provisioning, and selective SLA violation to maximize cloud provider revenue. Kashef et al. [50] propose a system architecture for cloud service providers that combines demand-oriented pricing with resource provisioning. They compare two revenue management techniques for cloud computing. The first sets the timing for offering price discounts, whereas the second determines the number of VMs that should be offered at full price. Anandasivam et al. [4] utilize a *bid price control* technique that originates from the revenue management literature for capacity management which accepts or denies incoming requests for service in order to increase revenue. Their model considers multiple resources such as CPU, memory, storage, and bandwidth. Xu and Li [134] present an infinite horizon stochastic dynamic program to maximize the cloud provider's revenue with stochastic demand arrivals and departures. They rely on dynamic pricing in revenue management framework as the main technique to maximize revenue.

There are a few studies in the literature that consider the problem of optimally allocating resources to different customers from different market segment. The problem of dynamically allocating resources to different spot markets for revenue maximization has been investigated by Zhang et al. [139]. Supply adjustment and dynamic pricing are used as a means to maximize revenue and meet customer demand. They model the problem as a constrained discrete-time and finite-horizon optimal control problem and adopt *Model Predictive Control* techniques to design the dynamic algorithm solution. Deciding on the optimal capacity segmentation for on-demand and spot market requests has been formulated as a Markov decision process by Wang et al. [127]. As a part of their work, they propose an optimal mechanism for spot market based on the uniform price auction. In their model, they only consider usage-based and demand-oriented pricing models and they do not consider subscription-based pricing.

There are a few other studies which they have roots in revenue management techniques and are relevant to the topic. Abhishek et al. [1] characterize the equilibrium where arriving requests can choose between fixed-rate usage-based and the demand-oriented pricing markets. Their theoretical and simulation based analysis suggest that fixed-rate pricing generates higher expected revenue than joint market (i.e., fixed-rate usage-based plus demand-oriented market). Wang et al. [126] investigate the problem of pricing in spot market to maximize the revenue of cloud providers. They argue that pricing at present impacts on the future. In order to characterize the impact of pricing on the present and future revenue, they present a demand curve model and formulate the problem as a time-average optimization problem. They propose two online algorithms to tackle the problem and their evaluation results show that both algorithms can obtain high revenue.

2.3.4 Demand-oriented

Demand-oriented pricing model previously discussed in Section 2.2.2. Therefore, we do not further discuss this model of dynamic pricing here.

We reviewed pricing and different aspects of it as a means of maximizing revenue for cloud providers in this section. The next section is dedicated to *cloud Federation* as another way of increasing revenue and profit for IaaS cloud providers. In broader sense, we first discuss on interconnected cloud environments including its benefit and challenges and then, we narrow down our focus to cloud federation and its economic aspects. Finally, We conclude by discussing on gaps and potential research areas in this regard.

2.4 Federated Cloud Environments

Recent studies show the benefits of interconnecting cloud environments and present attempts for the realization of federated cloud environment [14, 30, 99]. The benefits of interconnected cloud environments for both cloud providers and their customers are numerous and there are essential motivations for *cloud interoperability* which will eventually lead to the *Inter-Cloud*. Before discussing about benefits of cloud interoperability, we explore different feasible cloud interoperability scenarios.

2.4.1 Cloud Interoperability Scenarios

Cloud interoperability requires cloud providers to adopt and implement standard interfaces, protocols, formats and architectural components that facilitate collaboration. Without these *provider-centric* changes cloud interoperability is hard to achieve. Among different provider-centric approaches *Hybrid cloud*, *cloud federation* and *Inter-cloud* are the most prominent scenarios. A hybrid cloud allows a private cloud to form a partnership with a public cloud, enabling the *cloud bursting* application deployment model. Cloud bursting allows an application to run in a private data center and to burst into a public cloud when the demand for computing capacity spikes. Cloud federation allows providers to share their resources through federation regulations. In this paradigm, providers aim to overcome resource limitations in their local infrastructure, which may result in rejection of customer requests, by outsourcing requests to other members of the federation. Moreover, cloud federation allows providers operating at low utilization to lease part of their resources to other federation members in order to avoid wasting their non-storable compute resources. Last but not least is Inter-cloud, in which all clouds are globally interconnected, forming a worldwide cloud federation. Inter-cloud removes difficulties related to migration and supports dynamic scaling of applications across multiple clouds. Even if cloud interoperability is not supported by cloud providers, cloud customers are still able to benefit from *client-centric* interoperability facilitated by user-side libraries or third party brokers. *Multi-cloud* application deployment using adapter layer provides the flexibility to run applications on several clouds and reduces the difficulty in migrating applications across clouds. *Aggregated Service by Broker*, a third-party solution in this regard, offers an integrated service to users by coordinating access and utilization of multiple cloud resources. Figure 2.7 depicts the discussed classification and the remaining part of this section describe each scenario in detail.

Different combinations of Cloud providers (CPs) and Cloud customers give rise to a number of plausible scenarios between clouds [30]. Cloud customers can be either software/application *service providers* (SPs) who have their service consumers or *end users* who use the cloud computing services directly. SPs offer economically efficient services using hardware resources provisioned by cloud providers, i.e., cloud providers offer utility computing service required by other parties.

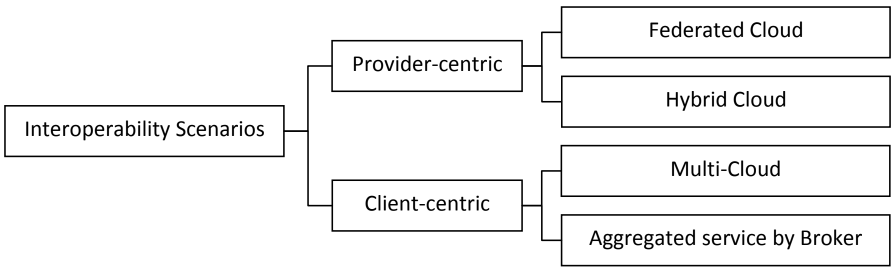


Figure 2.7: Cloud Interoperability Scenarios.

Federated Scenario

In this scenario, SP establishes a contract with CP that itself is a member of federation. A group of cloud providers are federated and trade their surplus resources amongst each other to gain economies of scale, efficient use of their assets, and expansion of their capabilities [19], e.g., to overcome resource limitation during spike in demands. In this model, the computing utility service is delivered to SP using resources of either one CP, or combination of different cloud providers. In such a scenario SP might be unaware of the federation and its contract is with a single cloud provider (Figure 2.8).

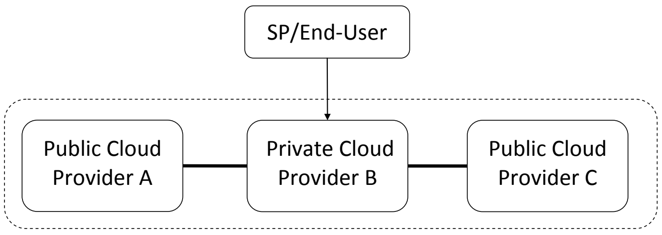


Figure 2.8: Federated cloud scenario.

Hybrid Cloud Scenario

In hybrid cloud architecture, an organization that owns its private cloud moves part of its operations to external CPs. The organization can also sell idle capacity to other providers during periods of low load. This extension of a private cloud to combine local resources with resources from remote CPs is called hybrid cloud. In this scenario, SP/end user application can scale out through both private and public clouds when the local infrastructure is insufficient. Furthermore, this scenario can be extended if the organization offers capacity from its private cloud to others when that capacity is not needed for internal operations (Figure 2.9).

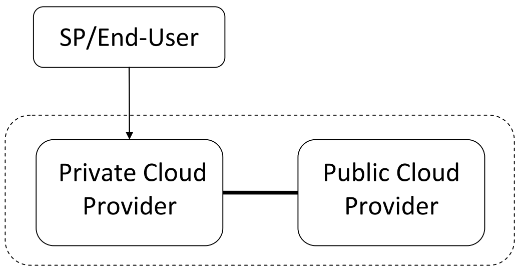


Figure 2.9: Hybrid cloud scenario.

Multi-Cloud Scenario

In this scenario, SP or end-users are responsible to manage resources across multiple clouds. Service deployment, negotiating with each CP, and monitoring each CP during service operation are performed by the SP or end-user applications. In this case, the SP may require using an adapter layer with different APIs to run services on different clouds or similarly end-user application needs a proper abstraction library. The important point about this scenario is that a separated layer handles all the issues regarding aggregation and integration of the clouds which is entirely apart from vendors and providers (Figure 2.10).

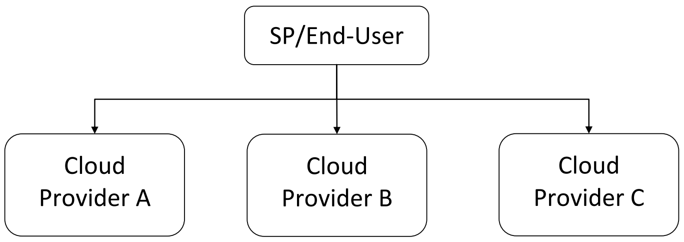


Figure 2.10: Multi-cloud scenario.

Aggregated Service by Broker

A new stakeholder, *broker*, aggregates services from multiple CPs and offers an integrated service to the SPs or end-users. The deployment and management of components has been abstracted by the third party broker. SPs or end-users benefit greatly from this model as the broker can provide a single entry point to multiple clouds. In this model, providers may also require to install some internal components to support aggregated services by a trusted broker (Figure 2.11).

2.4.2 Motivations for Cloud Interoperability

In this section, key benefits of cloud interoperability which provides essential motivations for interconnected cloud environments have been summarized (Figure 2.12).

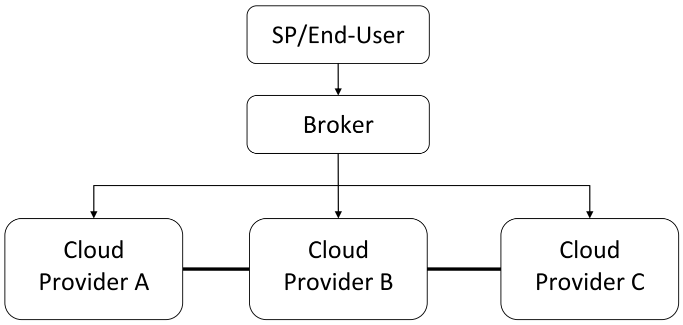


Figure 2.11: Aggregated service broker scenario.

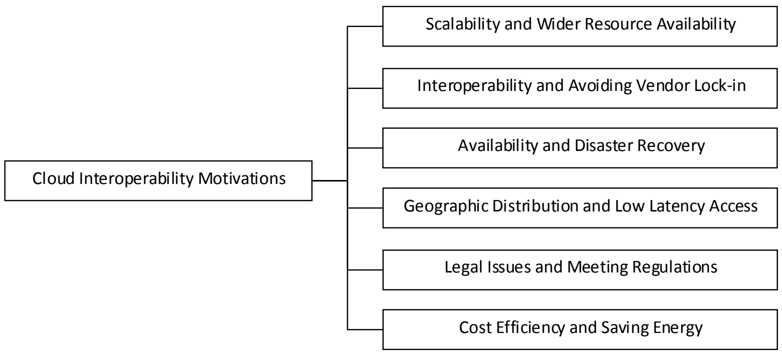

Cloud Interoperability Motivations
Scalability and Wider Resource Availability
Interoperability and Avoiding Vendor Lock-in
Availability and Disaster Recovery
Geographic Distribution and Low Latency Access
Legal Issues and Meeting Regulations
Cost Efficiency and Saving Energy

Figure 2.12: Cloud Interoperability Motivations

Scalability and Wider Resource Availability

Even though one of the key features of cloud computing is the illusion of infinite resources, capacity in cloud provider's data centers is limited and eventually can be fully utilized [6,17]. Growth in the scale of existing applications or surge in demand for a service may result in immediate need of additional capacity in the data center. Current service providers handle this issue by over-provisioning of data center capacity. That is, the average demand of the system is several times smaller than the capacity of their computing infrastructure. This strategy and the cost of its operation constitute a large expense for cloud owners. Actual usage patterns of many real-world application services vary with time and most of the time in unpredictable ways. Therefore, unexpected loads can potentially overburden a single cloud provider and lead to unreliable and interrupted services. It is overly restrictive in terms of small-size or private clouds. If cloud providers were able to dynamically scale up or down their data center capacity, they could save substantial amount of money and overcome the above issue. Scalable provisioning of application services under variable workload, resource, and network conditions is facilitated by interoperation of the clouds [14]. Cloud Federation helps the peak-load handling capacity of every enterprise cloud by resource sharing, without having the need to maintain or administer any additional computing nodes or servers [99].

One may argue that public cloud providers are outstandingly elastic, with the perception of unlimited resources, so providers never need immediate additional capacity in their data center and they never fit into the above scenario. However, this claim does not obviate the need for additional capacity by small size private clouds and for those applications requiring expansion across geographically distributed resources to meet Quality of Service (QoS) requirements of their users [14].

Interoperability and Avoiding Vendor Lock-in

In economics, vendor lock-in is a situation where a customer becomes dependent on a vendor for its products or services and cannot move to another vendor without considerable cost and technical effort. It is also perceived as one of the current drawbacks of cloud computing [7]. With respect to cloud computing, vendor lock-in is the direct result of the current difference between the individual vendor paradigms based on non-compatible underlying technologies, and the implicit lack of interoperability. Contemporary cloud technologies have not considered interoperability in design [10, 99]; hence, applications are usually restricted to a particular enterprise cloud or a cloud service provider. By means of cloud interoperability, cloud application deployment no longer needs to be customized. Cloud Interoperability makes cloud services capable of working together and also develops the ability of multiple clouds to support Cross-cloud applications [10].

Availability and Disaster Recovery

Although high availability is one of the fundamental design features for every cloud service, failure is inevitable. For instance, recently Amazon Web Services suffered an outage, and as a result, a group of large customers dependent on Amazon were affected seriously.[12] Unexpected failures can easily impose service interruption on a single cloud system. Aoyama and Sakai [6] look into an instance of a service failure in which a cloud system witnesses a natural disaster. They identify the most important requirements for disaster recovery through cloud federation. In order to enable cloud providers to continue the delivery of guaranteed service levels even in such cases, a flexible mechanism is needed to relocate resources among the multiple cloud systems. Moreover, highly-available cloud applications can be constructed by multiple cloud deployment to guarantee the required service quality, such as service availability and performance. Thus, cloud systems complement each other by mutually requesting required resources from their peers.

Geographic Distribution and Low Latency Access

It is highly unlikely that a single cloud provider owns data centers in all geographic locations of the world to meet the low-latency access requirement of applications. Moreover, existing systems do not support mechanisms to dynamically coordinate load distribution among different cloud data centers. Since predicting geographic distribution of users consuming a cloud provider's services is not trivial, the load coordination must happen automatically, and distribution of services must change in response to changes in the load [14]. Utilizing multiple clouds at the same time is the only solution for satisfying the requirements of the geographically-dispersed service consumers who require fast response time. Construction of a federated cloud computing environment is necessary to facilitate provisioning of such application services. Consistent meeting of the QoS targets of applications under variable load, resource and network conditions is possible in such an environment.

Legal Issues and Meeting Regulations

Many cloud customers have specific restrictions about the legal boundaries in which their data or application can be hosted [106]. Supplying resources in specific geographic loca-

[12] `https://cloudcomputing.sys-con.com/node/2416841`.

tions to meet regulations in place of those customers is an essential issue for a provider who wants to serve them. These regulations may be legal (e.g., an existing legislation specifying that public data must be in the geographic boundaries of a state or country) or defined by companies' internal policies [17]. Cloud interoperability provides an opportunity for the provider to identify another provider able to meet the regulations due to the location of its data center.

Cost Efficiency and Saving Energy

The usage-based *pay-as-you-go* pricing feature of cloud Computing directly awards economic benefits for customers by removing the cost of acquiring, provisioning, and operating their own infrastructures [7]. On the other hand, cloud computing providers should avoid the problem of the *idle capacity* (where their in-house hardware is not fully utilized all the time) and the problem of *peaks in demand* (where their own systems would be overloaded for a period). As the average demand of the system is several times smaller than the peak demand [7], providers are able to lease part of their resources to others, in order to avoid wasting their unused resources. Moreover, they can manage peaks in demand by purchasing resources form other underutilized providers. Both strategies help them to gain economies of scale, an efficient use of their asset and enlargement of their capabilities through enhanced resources utilization [19]. Furthermore, this cooperation among cloud providers lower the energy usage by promoting efficient utilization of the computing infrastructure.

In a study done by Le et al. [53], the plausibility of reducing cost and energy consumption by interconnecting clouds data centers has been investigated. They present a scenario in which a provider is able to save money by placing and migrating load across multiple geographically distributed data centers to take advantage of time-based differences in electricity prices. In addition, their policies reduce the required cooling power considering data centers located in areas with widely different outside temperatures. In general, a unified interface that provides federated interoperation between clouds would help providers saving cost and also reducing carbon footprint by energy-efficient utilization of physical resources.

2.4.3 Discussion

Among different incentives of interconnected cloud environments, scalability, wider resource availability, cost efficiency and saving energy advocates financial and economical benefits for cloud providers especially in federated cloud paradigm. From the cloud provider perspective, dynamic resource sharing among providers in federated cloud paradigm allows for overcoming the challenges of the load variability, future demand uncertainty, meeting regulations in place for those customers who have specific restrictions about the legal boundaries, low-latency access requirement of geographically-dispersed applications.

Once cloud providers are convinced that adoption of cloud cooperation and cloud interoperability awards them financial and economical benefits, the goal of ubiquitously interconnected clouds, i.e., Inter-cloud, is more likely to be achieved. We argue that cooperation between cloud providers cannot be achieved without the resolution of economic aspects. In this thesis, we look for techniques and methods to enhance cloud provider's revenue or profit; we, therefore, consider cloud federation as a means of achieving this purpose. To obtain the maximum benefit and given the complexity of federation decisions, it is important to address the issues regarding economic aspects of cloud federation which has got little attention in the literature. This requires addressing issues regarding novel methods of pricing suitable for interconnected cloud environments, and finally formation of Inter-cloud marketplaces. We dedicate the next section to economical challenges and components of the interconnected clouds specifically cloud federation which is the main focus of this thesis.

2.4.4 Economic challenges and enabling approaches

Cloud interoperability and federating clouds raises many more challenges than cloud computing both in terms of functional and non-functional aspects. To overcome these challenges, substantial efforts are required to research and develop tools and techniques in this regard. These challenges broadly cover security, Service Level Agreement (SLA), monitoring, virtualization, portability, economy, networking, provisioning, and autonomics. This is not our aim in this chapter to cover all the challenges in the interconnected cloud environments. We review four main economic-related challenges and identify possible enabling approaches. These four main challenges of federated cloud environments from the economic point of view are shown in Figure 2.13 and will be discussed in the followings.

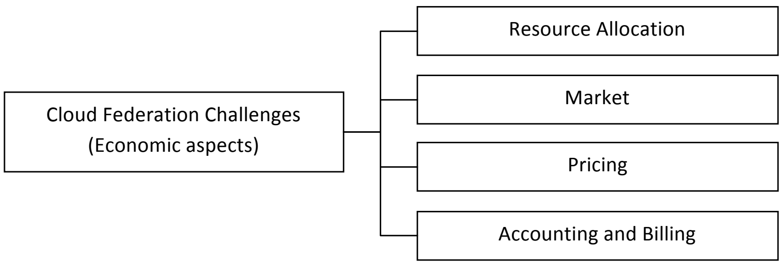


Figure 2.13: Main challenges of the federated cloud environment from the economic point of view.

Resource Allocation

Service selection in the customer′s side leads to resource allocation in the provider′s side. Resource allocation is a challenging issue from the cloud provider′s perspective. Cloud providers usually offer their virtualized resources based on different QoS levels, e.g., best effort and reserved. Physical resources in clouds are shared between cloud users. Therefore, allocation strategies are needed to allocate resources to the requests in a profitable manner while fulfilling requests′ QoS requirements.

As the number of resource consumers are increasing, clouds need to share their resources with each other to improve their quality of service. In general, such a collaborative cloud computing system (e.g., cloud federation) is prone to contention between user requests for accessing resources [103]. Contention happens when a user request cannot be admitted or cannot acquire sufficient resources because resources are occupied by other requests (e.g., requests from federated cloud provider). This issue is called *resource contention* in the literature.

Resource contention is not a new issue in federated environments. Various solutions have been proposed for the resource contention problem in federated cloud environments and other interconnected distributed computing systems [99, 103]. There is growing interest in the adoption of market-based approaches for allocation of shared resources in computational systems [73]. Mihailescu and Teo [73] propose a dynamic pricing scheme for federated sharing of computing resources, where federation participants provide and use resources. They show that in their proposed dynamic scheme, the user welfare, the percentage of successful requests, and the percentage of allocated resources increase in comparison to the fixed pricing [72]. Gomes et al. [40] propose and investigate the application of market-oriented mechanisms based on the *General Equilibrium Theory* to coordinate the sharing of resources between clouds in the federated cloud.

In a federated Cloud, providers can mutually collaborate to share their resources and fulfill the demand among each others. For instance, a provider can outsource requests to other providers when accommodating requests within local infrastructure is not possible for example in peak hours. Undoubtedly, this is the case when the expected revenue from these customers' requests is higher than the cost of outsourcing them. Therefore, the provider obtains higher profit because it can provide service for more customers without facing capital costs of acquiring IT equipment which might be costly. Similarly, a provider that has underutilized resources could lease part of them to other providers in the federation [35]. The profitability of a cloud provider in a federated scenario highly depends on a wide range of parameters such as the provider's incoming workload, the cost of outsourcing additional resources, the revenue of leasing underutilized capacity, or the cost of maintaining the provider's resources operative. Therefore, cloud providers require having a comprehensible understanding of the consequences of every decision they make.

Goiri et al. [35] propose an economic model that characterizes situations that assist decisions in a federated cloud, such as when to outsource resources to other providers, when to admit requests from other providers, and how much capacity to contribute to the federation. Samaan [104] addresses the problem of maximizing the IaaS cloud provider's long-term revenue in federation where current capacity sharing decisions depend on the revenue obtained from previous decisions. The uncertainty in future revenue has been taken into account as a participation incentive to sharing in the repeated game of VM outsourcing with the option of offering all underutilized capacity in the spot market. A set of self-enforceable cloud providers capacity sharing strategies has proposed that maximize the social welfare and yet can achieve more revenue than what each cloud provider can achieve individually without participating in the federation.

Similarly in Chapter 3 of this thesis, we propose a model for trading of cloud services based on competitive economic models. We consider circumstances in which cloud providers offer on-demand and spot VMs while they participate in a federation.[13] The resource manager unit evaluates the cost-benefit of outsourcing an on-demand request to a third party or allocating resources via termination of spot VMs. The ultimate objective is to increase the profit, to decrease the rejection rate, and have access to seemingly unlimited resources for on-demand requests.

Market

Interoperability between different providers allows cloud customers to use the service across clouds to improve scalability and reliability [71]. Computing as a utility can be considered as one of the main goals in federated cloud computing where resources in multiple cloud platforms are integrated in a single resource pool. A key challenge in this regard is how cloud providers interact with each other to realize collaboration [140].

A cloud provider is able to meet the peak in resource requirements by buying resources from other cloud providers. Similarly, when a cloud provider has idle resources, it can sell these resources to the federated cloud market. In order to enable such a resource sharing and collaboration among cloud providers, there is a need for a market-place with exchange facilities that helps providers in trading resources amongst each other [11].

Buyya et al. [14] proposed federated network of clouds mediated by a cloud exchange as a market maker to bring together cloud providers and customers. It supports trading of cloud services based on competitive economic models such as commodity markets and auctions. Blueprints for a comprehensive governance and marketplace architecture for interconnected cloud environments can be found in [11]. Federated cloud providers

[13]Spot VMs are VMs that can be terminated by providers whenever the current value for running such VMs (defined by the provider) exceeds the value that the client is willing to pay for using such resource.

require a clear understanding of the ramifications of each decision they make regarding selling/buying resources to/from other providers. Goiri et al. [35] present a plausible characterization of providers decisions operating in a federated cloud including outsourcing requests, or renting idle resources to other providers.

Market-based approaches for allocation of shared resources have proven their potential in computational systems [70]. To address the market-based resource allocation mechanism design problem, Mihailescu and Teo [73] propose a reverse auction-based mechanism. The market maker selects the sellers for allocation, based on the published price, such that the underlying resource costs are minimized. Afterwards, the actual payments for the winning sellers are determined based on the market supply.

Pricing

Pricing and profit are two important factors for cloud providers to remain in the business. Cloud federation allows providers to trade their resources under federation regulations. Strategies regarding selling and buying of resources in federated cloud environments are important issues that should be considered by providers. Providers need to profoundly consider how to price their services in the federated cloud market to assure profitability. In fact resource pricing, market mechanism, and resource allocation and provisioning in the federated cloud environment are correlated issues and these cannot be considered in isolation.

Dynamic resource pricing is a necessity in interconnected cloud environments where distributed cloud providers seek to accommodate more customers and meanwhile they compete with each other. Li et al. [57] design algorithms for inter-cloud resource trading and scheduling in a federation of geo-distributed clouds. For virtual machine trading among clouds, they apply a double auction based mechanism that is strategy-proof, individual rational, and ex-post budget balanced. Their proposed method optimally schedules stochastic job arrivals with different SLAs onto the VMs, and judiciously turns on and off servers based on the current electricity prices. Similarly, Mihailescu and Teo [73] argue that dynamic pricing is more suitable for federated sharing of computing resources, where participants may both provide and use resources. They present an auction framework that uses dynamic pricing to allocate shared resources. They show that using their proposed dynamic pricing scheme, the user welfare, the percentage of accepted requests, and the percentage of allocated resources increase in comparison to fixed pricing. In this thesis, we propose policies and a way to price resources in federated cloud. We also propose a financial option based cloud resources pricing model to help providers in federated cloud environments.

Accounting and Billing

In a federated cloud environment, accounting and billing must be carried out in a way that meets the requirements of federated cloud scenario. Some identified challenges may affect the design of the accounting and billing in this environment; the actual placement of the resources may not be known to the entire system, and may also change during the service lifetime. Moreover, the number of required resources composing a service can dynamically go up and down to cope with a change in demand [28]. Primarily, it is required that resource usage is monitored for billing and accounting purposes. Additionally, in federated cloud environments, cloud providers expect the federation to be honest in its accounting and billing practices [43].

Any accounting and billing approach must be performed in a fair and standardized way both (a) for cloud customers and cloud provider interactions; and (b) for cloud provider to cloud provider interactions [28]. Moreover, for billing, those approaches must take into account the postpaid and prepaid payment schemes for capacity that varies over time in response to customer requirements. Elmroth et al. [28] presented a solution for accounting and billing in a federated cloud environment. The focus of the work is in the design of the accounting and billing system, utilizing existing alternatives, for the RESERVOIR [99] project. They focused on accounting and billing between a cloud provider and customer (retail). Provider to provider accounting and billing (wholesale) still remains as an open issue and needs further considerations.

2.5 Thesis Scope and Positioning

This thesis investigates market and economics-inspired mechanisms to maximize IaaS cloud providers' income with limited resource available in data centers. Therefore, four techniques helping IaaS cloud providers to enhance their income have been proposed in this thesis. In the remaining part of this section, we present the scope of the current thesis and its positioning within the research area.

As stated earlier, there are two main stakeholders in cloud environments, cloud providers and customers. In comparison to studies devoted to minimizing cost for cloud customers, relatively much less attention has been given to techniques of maximizing providers' revenue. Therefore, in this thesis, we focus on benefits of IaaS cloud providers where they offer computational services in the form of Virtual Machine (VM) instances. The provider is faced with stochastic and dynamic arrivals and departures of customers who submit requests to acquire the VM instances. The cloud provider is not aware of applications that are executed inside a VM instance; and moreover the duration, that VM instances remain active in the system is not known in advance.

In this thesis, the main objective is to maximize revenue or profit of the provider. As the computational services offered by IaaS cloud providers are non-storable or perishable products, in order to maximize revenue, cloud providers must accept as many requests as possible to achieve the highest possible utilization. Nevertheless, they must guarantee Quality of Service (QoS) based on the agreed Service Level Agreement (SLA) with customers. The liability to provide required QoS according to the service level agreement (SLA) introduces a number of non-trivial trade-offs to IaaS providers with respect to revenue maximization. Moreover, even though one of the key features of cloud computing is the illusion of infinite resources, capacity in cloud provider's data centers are limited and eventually can be fully utilized. Therefore, to achieve the goal of revenue maximization, providers require efficient resource management strategies in addition of effective pricing and efficient market mechanisms. Such an efficient resource management aids in maximizing resource utilization and selecting more profitable requests while it would help in creating trust and goodwill among customers on the cloud service providers by lower number of QoS violation.

Table 2.1: The thesis scope

Characteristic	Thesis Scope
Target Party	IaaS Cloud Providers (Federated and non-federated)
Objective	Maximizing Revenue or Profit
Constraints	Data center capacity and SLA
Methodology	Market and economics-inspired mechanisms
Workload	Virtual Machine Requests

Methodologies proposed in this thesis are developed for two main situations: 1) when the provider acts solely using their in-house resources to serve customers and 2) when it participates in a cloud federation and benefits from outsourcing requests. Each core chapter of this thesis focuses on a different and original economics-inspired method of maximizing the cloud provider's income. Chapters 3 and 4 focus on cloud providers in federated cloud environments, while center of attention is on individual cloud provider's revenue maximization in Chapters 5, 6 and 7. The scope of this thesis is summarized in Table 2.1. The remainder of this section will be devoted to positioning of each core chapter of the thesis including gap analysis and comparison to the most related work. Table 2.2 summarizes details and the scope of each chapter.

Chapters 3 of this thesis addresses the decision making challenges of an IaaS cloud provider who dedicate part of the capacity to demand-oriented spot market for local customers while participating in a cloud federation. The chapter discusses the following questions: to what extent a provider must contribute to the federation, how much the provider should charge other providers for their service in federation (pricing), when it is beneficial to outsource a request to others or alternatively shrink the spot market size and run the request in-house. The most related study has been done by Goiri et al. [33, 35] where they present a profit-driven policy for decisions related to outsourcing or selling idle resources. On that approach, providers have the option of shutting down unused nodes of the data center to save power. However, they only consider single pricing channel (usage-based pay-as-you-go model) to customers while in common practice provider might sell virtual machine instances through different pricing channels such as usage-based and demand-oriented pricing models. Moreover, they propose a very simple discounting method for pricing in the federation which does not provide enough incentive for the self-interested cloud providers in federation to share their resources. Another related work is done by Samaan [104] which follows the same goal of maximizing the revenue by selling underutilized capacity to federation. The study tries to regulate capacity sharing in a federation of IaaS cloud providers in the presence of demand-oriented spot market. However, their approach differs from ours as they use game theory approach to address the issue and no specific pricing model is imposed for the federation.

In Chapter 4, we address the same problem as in Chapter 3 while we focus on the challenges regarding subscription-based pricing model and future contracts. A financial option-based market mechanism is proposed which allows for hedging against the critical and risky situation of imposing SLA violation when the cloud provider allocates underutilized capacity of data center reserved by the subscription-based customers to customers of other pricing channels, e.g., usage-based customers (overbooking). To the best of our knowledge, this is the first attempt to economically manage resource reservation and future contracts in federated cloud environments. The most relevant study is done by Meinl and Neumann [65] that analyzes the use of real option in a contract market, to manage resource reservation in Gird environments. They use option as a contract to perform reservation for time and budget sensitive customers. In their model, Grid customers want to minimize expenses, whereas Grid providers want to maximize their return on investment. We use option as a hedging mechanism for the overbooked reserved capacity to enhance provider profit while they focus on social welfare when there is risk of price fluctuation and the risk of not being allocated in the future. Besides, they consider advance reservation for Grid jobs with deadline and budget constraints, while our reservation model is inspired by subscription-based pricing model in which customers pay upfront fee to reserve VM instances for a period of time (e.g., one year) and utilize the reserved capacity as need arises.

In Chapter 5, we address the problem of maximizing revenue when all three common pricing models in IaaS marketplace, i.e., usage-based, subscription-based and demand-oriented pricing models are jointly supported by the provider. The main research question we address is: “with limited resources that are available, and considering the dy-

Table 2.2: The thesis scope

Chapter	Methodology	Characteristics
3	Demand-oriented Market and Federation	Allocation, Pricing, Market
4	Subscription-based Market and Federation	Allocation, Pricing, Market
5	Revenue Management	Capacity Control
6	Auction (Mechanism Design)	Truthful, Optimal, Fair
7	Prototyping	Pricing as a service

namic and stochastic nature of customers' demand, how expected revenue can be maximized through the optimal capacity allocation to customers from each pricing channel?". We frame our contributions within a *yield management* framework that incorporates capacity control techniques. Similarly, Wang et al. [127] consider the coexistence of multiple pricing channels that only focuses on usage-based and demand-oriented market. They formulate the problem as a *Markov Decision Process* model to make decision on the optimal capacity segmentation between requests from usage-based pay-as-you-go and demand-oriented markets. However, our work in Chapter 5 is the first attempt that has been made to incorporate all three pricing models in revenue maximization problem.

At present, the design of an efficient, fair, and revenue-maximizing auction mechanism for demand-oriented pricing of cloud computing resources is an open research challenge, and of great interest to cloud providers. Amazon Web Services (AWS) offers an auction like approach to sell their computational resources; however, they have revealed no detailed information regarding their mechanism. In contrast to Amazon's claim, several studies suggest that it is unlikely that prices in the spot market to be fully set according to market supply and demand [9, 134]. Moreover, there is doubts about the efficiency and truthfulness of the mechanism used by AWS [113]. Therefore, in Chapter 6 of this thesis, we aim at design of an auction mechanism for selling the spare capacity available in cloud data which maximizes the profit while it is *fair (envy-free)* and *truthful*. There are a few studies that try to design auction mechanism suitable for demand-oriented pricing of IaaS cloud marketplaces [128, 134, 137]. They are different in terms of goal of auction design such as truthfulness, revenue maximization, fairness or characteristics such as single item or multi-item, single sided or double-sided. The most relevant work has been done by Wang et al. [127] in which they propose optimal recurrent auction for a spot market. Their work differs from ours since they adopt a *Bayesian optimal* auction design wherein it is assumed that the customers' private values are drawn from a known distribution while we focus on *prior free* mechanism design when no knowledge about the distribution of values is available a priori.

Chapter 7 introduces a dynamic pricing prototype system based on the proposed demand-oriented pricing in Chapter 6. The proposed system is an implementation of an open source framework called Spot instance pricing as a Service (SipaaS). SipaaS provides a set of web services to facilitate running a spot market for IaaS cloud platforms. In order to make a real spot market environment to sell computational cloud resources, the chapter presents an extension to *OpenStack* – an open source platform for building private and public clouds. Accordingly, *Horizon* – the OpenStack dashboard project – is extended to make use of the SipaaS framework. To the best of our knowledge, this is the first attempt to create a spot market environment in OpenStack. IaaS cloud providers can run spot market resembling the Amazon EC2 spot instances using our proposed spot instance pricing as a service framework.

2.6 Summary

This chapter described the concepts, background, and market and economics-inspired methodologies of maximizing IaaS cloud providers' revenue. We have investigated and classified different pricing and market design methodologies of maximizing profit in cloud environments. We reviewed the state-of the-art developments related to each topic. This helped us to identify the gap and research direction and what has to be done to address profit maximization problem of IaaS cloud providers. We discussed two main different situations: 1) when the provider acts solely using their in-house resources to serve customers and 2) when it participates in a cloud federation and benefits from outsourcing requests. We also reviewed the motivations and challenges regarding economic aspects of cloud federation. This chapter has finally concluded with a discussion of the scope and positioning of each core chapter of the thesis. We have analyzed open research challenges in this regard and positioned our proposed economics-inspired methods and techniques within this research area.

Part I

Profit Maximization in Federated Cloud Environments

Introduction to Part I

CLOUD federation is the practice of interconnecting cloud providers to share and trade resources. It allows cloud providers to share their underutilized capacity during low-demand periods and borrow capacity during peaks to maximize their profit and enhance their customers' experience. Interconnecting clouds raises many technical, functional, and non-functional challenges. The first part of this thesis is dedicated to the study of economic implications of federated cloud environments. In particular, we investigate the design and development of market mechanisms that provide incentives for cloud providers to join the cloud federation by addressing the profit maximization problem. We also propose economics-inspired resource allocation methods helping cloud providers to make strategic decisions in such a collaborative while competitive environment. Chapter 3 proposes policies to increase utilization and profit for a cloud provider in a federation exchange market that help in the decision-making process of resource allocation. Chapter 4 presents a financial option-based market designed for future trades in a federated cloud marketplace.

This page intentionally left blank.

Chapter 3

Resource Provisioning Policies to Increase Profit

Cloud Federation is a recent paradigm that helps Infrastructure as a Service (IaaS) providers to overcome resource limitation during spikes in demand for Virtual Machines (VMs) by outsourcing requests to other federation members. IaaS providers also have the option of terminating spot VMs, i.e, cheaper VMs that can be cancelled to free resources for more profitable VM requests. By both approaches, providers can expect to reject fewer profitable requests. For IaaS providers, pricing and profit are two important factors, in addition to maintaining a high Quality of Service (QoS) and utilization of their resources to remain in the business. For this, a clear understanding of the usage patterns, types of requests, and infrastructure costs is necessary while making decisions to terminate spot VMs, outsourcing or contributing to the federation. In this chapter, we propose policies for decision-making process to increase resources' utilization and profit. Simulation results indicate that the proposed policies enhance the profit, utilization, and QoS (smaller number of rejected VM requests) in a Cloud federation environment.

3.1 Introduction

IN recent years, Cloud Computing [7, 15, 123] has become a consolidated paradigm for delivery of services through on-demand provisioning of virtualized resources. By the emergence of this paradigm, along with support of companies like Amazon, Microsoft, and IBM, the long envisioned dream of computing as a utility finally has come true. Now customers are able to use resources and services in a pay-as-you-go manner from anywhere and at anytime. Among the different methods to deliver Cloud services, Infrastructure as a Service (IaaS) allows Cloud provider to sell resources in the form of Virtual Machines (VMs) to customers.

One of the key motivations for IaaS providers is the possibility of making profit by leveraging their available data center resources to serve potentially thousands of users. Therefore, Cloud providers aspire to accept as many new requests as possible with the main objective of maximizing profit; nevertheless, they must guarantee Quality of Service (QoS) based on the agreed Service Level Agreement (SLA) with customers. Achieving this goal requires efficient resource management strategies.

To be able to offer QoS guarantees without limiting the number of accepted requests, providers must be able to dynamically increase the available resources to serve requests. One possible source for additional resources is idle resources from other providers. In order to enable such a scenario, coordination between providers has to be achieved, possibly through establishment of a Cloud federation [14,51,99].

A Cloud federation allows providers to trade their resources through federation regulations. In this paradigm, providers aim to overcome resource limitation in their local infrastructure which may result in rejection of customer requests, by outsourcing requests to other members of the federation. Moreover, Cloud federation allows underutilized providers to lease part of their resources to other members of the federation, usually at cheaper prices, in order to avoid wasting their *non-storable* (*perishable*) compute resources. Both cases lead to the enhancement in profit and elasticity for providers, if this opportunity is properly used. By this we mean that providers should make an intelligent decision about utilization of the federation (either as a contributor or as a consumer of resources) depending on different conditions that they might face.

A challenging condition for providers occurs when they dedicate part of their capacity in the form of spot VMs. Spot VMs are VMs that can be terminated by providers whenever the current value for running such VMs (defined by the provider) exceeds the value that the client is willing to pay for using such resources, as in the case of Amazon EC2 spot instances [120]. This type of VMs can be provided to users at a lower cost than on-demand VMs, usually in the spot market which works based on supply and demand. Existence of spot VMs certainly benefits IaaS Cloud providers, because spot VMs help them in making profit by increasing the utilization of the data center while waiting for incoming on-demand requests. When a federated cloud provider receives an on-demand request for VMs but has no idle resources within the data center, it has to decide between either increasing the spot price and terminating spot VMs, or outsourcing the request to another federation member.

Decision on outsourcing requests or renting part of idle resources to other providers is a complex problem that has been surveyed by several studies [34, 56]. To the best of our knowledge, the work in this chapter is the first attempt to incorporate the outsourcing problem with the option of terminating spot VMs within a data center. Our main objective is to maximize a provider's profit, by accommodating as many on-demand requests as possible. Our main contribution is to propose policies that help making decisions when providers have different choices regarding incoming requests: rejecting, outsourcing, or terminating spot leases to free resources for more profitable requests[1].

The remainder of this chapter is organized as follows: Related work is reviewed in section 3.2. In section 3.3, we define the system model, including customers and providers interaction. Section 3.4 describes the proposed policies and formalizes the decision equations. Evaluation and experimental environment are presented in section 3.5. Finally, we conclude this chapter in section 3.6.

3.2 Related Work

Despite several recently proposed platforms for Cloud federation [14, 99, 101], with different motivations and incentives for parties to join it, many fundamental problems and questions about federation remain unanswered. One of these problems is deciding when providers should outsource their local requests to other participants of the federation or how many and at what price they should provide resources to the federation. The outsourcing problem is not considered only in the context of federated clouds; it is also investigated as a way of increasing capacity or scalability of applications in hybrid clouds [27], distributed grid environment [52], and clusters [26].

Goiri et al. [34] present a profit-driven policy for decisions related to outsourcing or selling idle resources. On that approach, providers have the option of shutting down

[1] In this chapter, terms VM and resource are used interchangeably.

unused nodes of the data center to save power. However, they did not take into account different types of VMs (e.g. on-demand and spot) and possible actions like terminating low priority leases.

A user satisfaction-oriented scheduling algorithm for service requests is proposed by Lee et al. [55]. Such an algorithm tries to maximize cloud providers' profit by accepting as many service requests as possible, as long as QoS is kept at a certain level. In this regard, contracting with other service providers was taken into account as a method to avoid rejection of user requests. One of the main differences between this and our approach is that we specifically focus on federation of IaaS providers that serve requests for VMs. Moreover, we claim that federation, rather than being merely a technique for avoiding service rejection at the provider level, can also be a source of profit for providers by allowing them to negotiate on otherwise wasted resources at competitive prices.

The problem of how to value resources and how price may impact utilization is not trivial. Current public cloud service providers like Amazon, GoGrid and RackSpace usually adopt fixed pricing strategies for the infrastructure services they provide. However, fixed pricing models are not suitable for federated environments as a policy to be applied between its participants, because it neither reflects current market price of resources due to dynamism in supply and demand nor generates any incentives for providers to join the federation. Dynamic pricing of resources, however, lies outside the scope of this chapter, and has been a subject of other studies [71]. Hence, in this chapter, a policy based on the provider utilization is applied by federated providers to dynamically value resources.

The subject of leveraging spot VMs has recently attracted considerable attention. Andrzejak et al. [5] have proposed a probabilistic decision model to help users decide how much to bid for a certain spot instance type in order to meet a certain monetary budget or a deadline. Yi et al. [135] proposed a method to reduce monetary costs of computations using Amazon EC2s spot instances for resource provisioning. These works consider methods for increasing customers' benefit in using spot VMs, while we are interested in better resource provisioning policies for providers in the presence of spot VMs. Moreover, the problem of dynamic allocation of data center resources to different spot markets to maximize cloud provider's total revenue has been investigated by Zhang et al. [139].

A few works consider the application of market-oriented mechanisms in federated environments [40, 112]. These mechanisms mostly promote fairness and ensure mutual benefits for parties involved in the federation. Study and development of such techniques motivate both resource providers and resource consumers to join and stay in the federation market. Niyato et al. [83] study the cooperative behavior of multiple cloud providers and propose a cooperative game model. Our work, on the other hand, is focused on specific policies to be applied by cloud IaaS resource providers to decide when to buy computational resources and how resources should be made available in the market for other IaaS providers.

3.3 System Model

In this section, firstly, we describe the interaction between customers and providers including various types of services. Afterward, the scenario, assumptions, and requirements for cloud federation are discussed.

3.3.1 Interaction between customers and providers

Cloud computing providers, specifically IaaS providers, offer different types of VMs with different QoS and pricing models that help them support different types of applications and fulfill customers' requirements. This variety of QoS and pricing models also gives them more flexibility in resource management and to increase utilization. For example, Amazon Elastic Compute Cloud (EC2) [120] offers three different pricing models, *on-demand*, *reserved* and *spot*.

In this work, we consider the scenario where providers support two different levels of QoS and pricing models which are mostly based on Amazon pricing models. By providers, we mean the set of autonomous IaaS cloud providers who own a data center and serve a number of customers. Customers are either private users or other SaaS and PaaS providers, who submit requests to IaaS providers for instantiating VMs in either *on-demand* or *spot* types, based on their requirements.

On-demand VMs allow customers to pay for compute capacity by its hourly usage without a long-term commitment. Customers request for VMs, which are provisioned to them if the provider possesses enough resources, otherwise the request is rejected. After instantiation of VMs, customers can retain machines as long as they need them.

Spot VMs allow customers to reduce the cost of using VMs by accepting the risk of being canceled in favor of customers willing to pay more for the same resources. In this model, customers submit a spot VM request, including the number of spot VMs they want to instantiate, and the maximum price they are willing to pay per VM/hour, that is called bid. If the bid exceeds the current price, the request is served and VMs are instantiated. Otherwise, no VM is launched and the request remains in a pending state in the queue until the spot price goes below the bid price. VMs will run until either the customer decides to terminate them or the price goes above the bid.

The provider charges the customer based on the current price, which is calculated based on the minimum bid of running VMs in the system in this chpater (*uniform price auction*) [139]. There is a correlation between resource availability and current price. In case of resource shortage, the provider terminates VMs of low bid and replaces them with higher bid VM requests or on-demand requests. Consequently, bidding at higher price decreases the likelihood of VM termination by the provider.

Here, we consider two types of spot VM requests: *one-time* and *persistent*. One-time are spot requests that are not restarted after termination by providers, whereas persistent are spot requests that are kept in the data center to be re-executed until completed. Providers automatically instantiate new VMs for the persistent request each time the current spot price goes back below the bidding price.

The difference between on-demand and spot lies only in the guarantee about resources availability. Other QoS characteristics of VMs (such as memory and CPU power) are the same for both models and they are enforced by providers. Moreover, we assume that the user request has to be entirely served in the same data center.

In this study, we assume that infrastructure providers commit the actual amount of resources required by VMs, regardless of the actual users' usage pattern. This means that the resource manager does not apply methods of consolidation to increase the capacity of the data center [31, 34]. For instance, if two VMs requiring 1 unit of processing (e.g.

EC2 compute unit[2]) and they are running on the same physical node with two units of processing, the resource manager will not initialize another VM on that node, even if VMs are not using the total computing power allocated to them.

Considering that over-subscribing of resources may lead to violations in SLAs, providers have to use other methods to serve new on-demand requests if they are fully utilized. One alternative is increasing the spot VM price, which leads to cancellation of part of spot VMs and makes more room for on-demand requests. Another alternative is acquiring resources from other cloud providers that can be used to serve new on-demand requests. To make this scenario possible, it is important that providers engage in a federation so that they sell idle resources to other federation members at lower price than the customer's price. In exchange, they are also able to buy resources from other members when the demand increases for their resources. Interaction between federation members is detailed next.

3.3.2 Cloud Federation

The cloud federation scenario used in this chapter is presented in Figure 3.1, which is mostly inspired by the InterCloud project [14]. Each provider is autonomous, and has its own customers. Federation can help providers to absorb overloads due to spikes in demand. At the center of this model, the *Cloud Exchange* service plays the role of information service directory. With the aim of finding available resources from the members of federation, providers send an inquiry to the Cloud Exchange Service in case of shortage of local resources. The Cloud Exchange is responsible for generating a list of providers with corresponding service prices that can handle the current request. Therefore, the price list is used by providers to find suitable providers where requests can be redirected to.

Decision on allocating additional resources from a federated cloud provider is performed by a component called *Cloud Coordinator*. The amount of idle capacity that each provider shares with other members and the way providers price their resources are also decided by the Cloud Coordinator. These decisions significantly affect the profit

[2]1 EC2 Compute unit (ECU) is equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

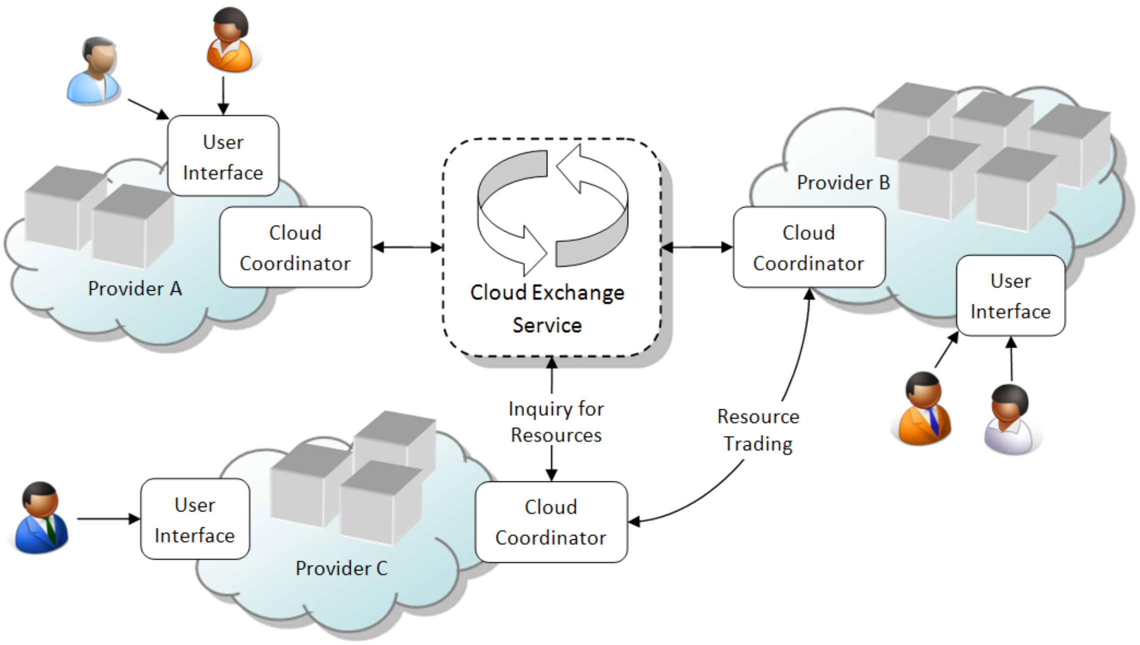


Figure 3.1: Cloud Federation Architecture.

of providers, and thus they are of paramount importance for the successful adoption of the federation paradigm by cloud providers. Moreover, agreements between federation members are necessary in order to make the federation profitable to all its members. We call these agreements *Federation Level Agreement* (FLA).

In this study, FLA requires that each provider dynamically prices its contributed resources (VMs) based on the idle capacity of its data center. Therefore, the instant federation price of a resource per hour can be computed as

$$F = \frac{M_p - M_{idle}}{M_p} \cdot (F_{max} - F_{min}) + F_{min} \quad , \tag{3.1}$$

where F is the resource's federation price; M_p and M_{idle} are total capacity and idle capacity of the provider's data center respectively, F_{max} is the on-demand VM price to customers and F_{min} is the minimum profitable price for the provider (*reserve price*). The provider does not sell resources for prices smaller than F_{min}. We discuss later in Section 3.5 about F_{min} that is anything larger than the price that compensates costs of keeping nodes of the data center up. This pricing mechanism facilitates load balancing between federated providers, since it results in cheaper price for providers with larger amount of resources.

A relevant issue about this mechanism is whether it reveals sensitive information about the provider, such as a provider's actual resource utilization (which might lead to inferences about a provider's revenue). Equation 3.1 does not reveal such sensitive information, since providers are free to advertise only a subset of their resources, and thus members cannot determine the overall utilization of other member's resources.

Considering the above scenario and assumptions, various policies are proposed in the next section to investigate how different decisions made by providers affect their profit and reputation with customers, when they provide the mentioned types of services.

3.4 Proposed Policies

A cloud provider may receive a request for on-demand VMs from its customer. Due to unavailability of resources, this request may not be served locally without violating QoS for other running VMs. Under such circumstances, the provider might decide to cancel a sufficient number of spot VMs (lowest bid first) to be able to accommodate more profitable on-demand requests. Also providers may look up to the federation, which provides an opportunity for outsourcing local requests to other members. In this case, an on-demand request received by a Provider A is actually served with resources from a Provider B. Provider B charges Provider A at the federation cost as given in Equation 3.1, which is typically lower than the prices that both Provider A and Provider B would charge the on-demand customers.

We propose policies that help the provider to increase profit, resource utilization, and user satisfaction, when providers are federation members and benefit from outsourcing requests, and also they are able to terminate spot VMs for serving on-demand requests. These policies only address the possibility of outsourcing on-demand requests. Outsourcing spot requests is not considered here, since the proposed policies are not designed to handle highly fluctuating prices of spot VMs.

To this end, we describe the proposed policies, which differ in how they handle new on-demand requests when they cannot be served by available local resources.

Non-Federated Totally In-house (NFTI)

In this policy, firstly providers consider termination of spot VMs with lower bids to accommodate a more profitable on-demand request. If this action does not release enough resources for the new on-demand request, it is rejected. This policy is considered as a baseline policy in order to allow verification of maximum profit a provider can make without the federation.

Federation-Aware Outsourcing Oriented (FAOO)

In this policy, each fully utilized provider firstly checks the Cloud exchange service for available offers by other members. Then, it outsources the request to the provider that offers the cheapest price. If outsourcing is not possible, Spot VMs are terminated as a last resort to accommodate the new on-demand request. This policy is considered to show whether always outsourcing is a profitable decision when fully utilized providers receive an on-demand request and spot termination is also possible.

Federation-Aware Profit Oriented (FAPO)

This policy compares the profit of outsourcing with termination of spot VMs. The idea behind this algorithm is that, in one hand, termination of spot VMs result in profit loss, and on the other hand, replacing spot VMs by on-demand ones increase the profit. Moreover, termination of spot VMs may result in spot price increase. In this policy, decisions are made based on $P(t)$, the *instant profit* of the provider at time t , which is given by

$$P(t) = R(t) - C(t) \ , \tag{3.2}$$

where $R(t)$ and $C(t)$ are revenue and cost at time t, respectively. $R(t)$ can be obtained as follow:

$$R(t) = R_o(t) + R_s(t) + R_{fed}(t) + R_{out}(t) \ , \tag{3.3}$$

where $R_o(t)$, $R_s(t)$, $R_{fed}(t)$ and $R_{out}(t)$ are revenue of on-demand, spot, contributed to federation and outsourced resources, respectively. By contributed to federation resources, we mean those local resources used by other members of the federation to serve their customers. $R_o(t)$ and $R_{out}(t)$ are calculated based on the following equations:

$$R_o(t) = vm_o(t) \cdot F_o \ , \tag{3.4}$$

$$R_{out}(t) = vm_{out}(t) \cdot F_o \ , \tag{3.5}$$

where F_o is the on-demand resource price per resource per hour, which is a constant value for all providers, and $vm_o(t)$ and $vm_{out}(t)$ are the number of on-demand VMs running locally and outsourced VMs, respectively. $R_s(t)$ is given by

$$R_s(t) = vm_s(t) \cdot F_s(t) \ , \tag{3.6}$$

where $vm_s(t)$ is the number of running spot VMs and $F_s(t)$ is the price of the spot VMs at time t. $R_{fed}(t)$ is calculated based on the summation of all VMs contributed to federation according to the following equation:

$$R_{fed}(t) = \sum_{i=1}^{vm_{fed}(t)} F_{fed_i} \ , \tag{3.7}$$

where $vm_{fed}(t)$ is the number of VMs contributed to federation and F_{fed_i} is the associated price for each VM contributed to federation.

In order to determine $C(t)$ in Equation 3.2, both operational cost and cost of outsourcing are considered. Therefore, $C(t)$ is given by

$$C(t) = C_p(t) + C_{out}(t) \ , \tag{3.8}$$

where $C_P(t)$ is the operational cost, which includes cost of acquiring and operating the data center servers (i.e, hardware and software acquisition, staff salary, power consumption, cooling costs, physical space, amortization of facilities, etc). $C_{out}(t)$ is the cost of outsourced VMs that a provider pays to federation members hosting its requests. Without loss of generality, we can assume that a constant value for $C_p(t)$ represents various combination of the costs and effect of change in constituting parameters. Further analysis of the cost in the constituting parameters on $C_p(t)$ is left as a future analytical work. Here, a constant value for $C_p(t)$ is assumed, whereas $C_{out}(t)$ depends on the renting price of the outsourced VMs. $C_{out}(t)$ is given by

$$C_{out}(t) = \sum_{i=1}^{vm_{out}(t)} F_{out_i} \ , \tag{3.9}$$

where F_{out_i} is the amount paid for each outsourced VM vm_i.

By putting all the above equations together, $P(t)$ is calculated as follows:

$$\begin{aligned} P(t) =& vm_o(t) \cdot F_o + vm_s(t) \cdot F_s(t) + \sum_{i=1}^{vm_{fed}(t)} F_{fed_i} \\ &+ vm_{out}(t) \cdot F_o - C_p(t) - \sum_{i=1}^{vm_{out}(t)} F_{out_i}. \end{aligned} \tag{3.10}$$

Considering Equation 3.10, *FAPO* policy has two choices in order to improve the provider's profit. When a request for n on-demand VMs arrives at time t, and local infrastructure can only accommodate m VMs $(m < n)$, it can either decide to terminate the $n - m$ lowest bid spot VMs or outsource the new request. To choose the best approach, *FAPO* policy estimates the instant profit in future time t' for both approaches $(t' = t + \varepsilon,\ \varepsilon \to 0)$.

The spot price is always set to the value of the lowest bid being served. Therefore, termination of the $n - m$ lowest bid spot VMs may increase spot price from $F_s(t)$ to $F_s(t')$ $(F_s(t) \geq F_s(t'))$. But, according to the proposed model, this increment only affects those spot requests whose accounting period expires on or after t', because the price of a spot VM is set at the beginning of each accounting period, which is one hour. Assuming k as the number of spot VMs whose accounting period start on t', the estimated instant profit of terminating $(n - m)$ spot VMs and accommodating n on-demands VMs locally, $P_1(t')$, is computed as:

$$\begin{aligned} P_1(t') = {} & (vm_o(t) + n) \cdot F_o + vm_{out}(t) \cdot F_o - C_p(t) \\ & + (vm_s(t) - (n - m) - k) \cdot F_s(t) + k \cdot F_s(t') \\ & + \sum_{i=1}^{vm_{fed}(t)} F_{fed_i} - \sum_{i=1}^{vm_{out}(t)} F_{out_i}. \end{aligned} \tag{3.11}$$

In the case of outsourcing, estimated instant profit at time t', $P_2(t')$, is defined as:

$$\begin{aligned} P_2(t') = {} & vm_o(t) \cdot F_o + vm_{out}(t) \cdot F_o \\ & + n \cdot F_o - C_p(t) + vm_s(t) \cdot F_s(t) \\ & + \sum_{i=1}^{vm_{fed}(t)} F_{fed_i} - \sum_{i=1}^{vm_{out}(t)} F_{out_i} - n \cdot F_{offer} \ , \end{aligned} \tag{3.12}$$

where F_{offer} is the lowest offered price in the federation. The policy compares $P_1(t')$ and $P_2(t')$ to make its decision. Therefore, similar terms can be eliminated from both equations and they are calculated as follows:

$$P_1(t') - P_2(t') = k \cdot F_s(t') - (n - m + k) \cdot F_s(t) + n \cdot F_{offer}. \tag{3.13}$$

Consequently, the policy decides to outsource requests when $P_1(t') - P_2(t') < 0$; otherwise, if $P_1(t') - P_2(t') \geq 0$, termination of spot VMs is more profitable and it will be recommended by the policy.

It is worth noting that none of the policies takes into account the duration of the request (request lifetime), because the provider does not have information about how long current VMs will remain in the system. Strategies that consider prediction of future resource availability to drive decisions can be explored as future works.

3.5 Evaluation

This section presents an evaluation of the policies presented in the previous section. First, we describe simulation settings and performance metrics, and then experimental results are presented and discussed.

3.5.1 Experimental Settings

The experiments presented in this section were developed using CloudSim [16] discrete-event cloud simulator. The simulated cloud scenario is composed of a federation containing multiple IaaS cloud providers. The number of providers is one of the simulation parameters, and we evaluate the effect of the policies considering different numbers of federation members.

For the sake of simplicity, we assume only one type of VM is offered by providers. The VM configuration is inspired by Amazon EC2 small instances (1 CPU core, 1.7 GB RAM, 1 EC2 Compute Unit, and 160 GB of local storage). Adding different types to the model can be considered as an extension of the current work.

Providers follow the pricing of Amazon Web Services (AWS) [120] at the time of experiments. That is, all providers charge their customers based on hourly usage, at the cost of \$0.085 per hour per on-demand VM. In the case of spot VMs, the provider charges customers based on the spot price, which fluctuates periodically according to the minimum bid in the system and resource availability. The price for each spot VM is set at the beginning of each VM-hour for the entire hour.

On the customer side, for the purpose of the bidding algorithm of spot requests, we assume that customers do not bid higher than the on-demand price, because higher bids behave like on-demand requests (they are never canceled) but generate even higher profit for the providers. Our bidding algorithm generates a uniformly-distributed random value between the minimum of bid \$0.020 and maximum of \$0.085. The minimum price is set in such a way that the value offered by customers is still enough to cover operational costs of serving the request, even though it might result in no profit for the provider. We also used \$0.020 and \$0.085 for F_{min} and F_{max} respectively in Equation 3.1 for pricing resources contributed by each provider in the federation.

Each simulated data center contains 128 servers, and each server supports 8 VMs. So, each provider is able to concurrently host 1024 VMs. We assumed that operational costs are constant and the same for all the providers, so they are not considered in the experiment.

For the sake of accuracy, each experiment is carried out 20 times by using different workloads and the average of the results is reported. We explain the workload setup details in the following subsection.

3.5.2 Workload setup

Due to lack of publicly available workload models and real traces of IaaS clouds, we apply a nine day long workload generated by the Lublin workload model [59]. To adapt this model to our scenario, we consider the number of nodes in each Lublin request as the number of VMs of the request, and this number is limited to 32 simultaneous VMs for each request. The first 12 hours of simulation and the last 36 hours are warm-up and cool-down periods respectively, and they are discarded from results. Therefore, a one week long period of simulation is considered.

The only parameters of the Lublin workload model changed for these experiments are job runtime parameters. We changed the first parameter of the Gamma distribution for runtime in Lublin model from the default value of 4.2 to 7.2 in order to generate longer VM requests. The user workload submitted to each provider is generated based on the above configuration. Providers are equidistantly distributed among time zones. Because the generated load has a daily cycle with peak hours, providers' loads vary. For example, while provider A is at its peak period, provider B will be at its off-peak time.

We intend to study the behavior of different policies in different situations. For this purpose, effects of four input parameters are investigated.

The first input parameter is the system load. The difference in the proposed policies lies on the action taken at the time at which a provider's resources are fully utilized and requires additional resources. In this direction, the arrival rate of requests has been selected as the most suitable parameter to adjust the load of a provider. With the intention of changing providers load, arrival rate of requests has been changed by varying the *aarr* parameter of the model between 8.2 and 6.4. *aarr* is the shape parameter of the Gamma distribution.

Another parameter that impacts the policies' performance is the ratio of spot requests to total requests (on-demand plus spot requests). We named this parameter β ($0 \leqslant \beta \leqslant 100\%$), and this is defined as follows. After generating the 9 days long workload, we randomly select some of the generated requests as spot requests, so β percent of the requests becomes spot requests.

The third parameter evaluated, α, contains the rate of spot requests that are persistent. It impacts the provider's profit because it determines the amount of requests that are going to be kept in the provider to be served when resources become available.

Finally, *Number of providers* is another parameter that is considered. This parameter is important because it increases the chance of members finding other members with lower load to select as the target of outsourced requests.

3.5.3 Performance Metrics

We applied the following metrics to analyze the impact of the proposed policies in the providers:

1. *Profit.* This metric is calculated for each provider and is defined as the achieved revenue during a time period minus the cost incurred in the same time period. In our results, we ignore operational costs. Therefore,

$$Profit(\Delta t) = Revenue(\Delta t) - Cost_{out}(\Delta t) \quad , \tag{3.14}$$

   where $Revenue(\Delta t)$ is the revenue obtained during Δt including on-demand, spot, contributed to the federation, and outsourced requests, whereas $Cost_{out}(\Delta t)$ is the cost of the outsourcing VMs at the same period.

2. *Utilization.* This metric is defined as the ratio between the number of hours of VMs used by requests (both local and contributed to the federation) and the maximum number of hours of VMs in a time period.

$$Utilization(\Delta t) = \frac{\sum_{i=1}^{vm} runtime(vm_i)}{vm_{max} \cdot \Delta t} \quad , \tag{3.15}$$

   where vm is the total number of VMs including on-demand, spot and contributed to federation VMs and vm_{max} is the maximum number of VMs that a provider can run simultaneously in its data center. $runtime(vm_i)$ shows the corresponding runtime for each VM.

3. *Number of rejected on-demand VMs.* This metric shows the number of on-demand VMs rejected. It considers only on-demand requests, because providers never reject spot VM requests which are kept in waiting queues until the bid price is reached. Note that, this metric does not show the number of rejected requests because each request may contain demand for more than one VM. We select this metric instead of the rejected number of requests because it better shows potential revenue losses.

3.5.4 Results

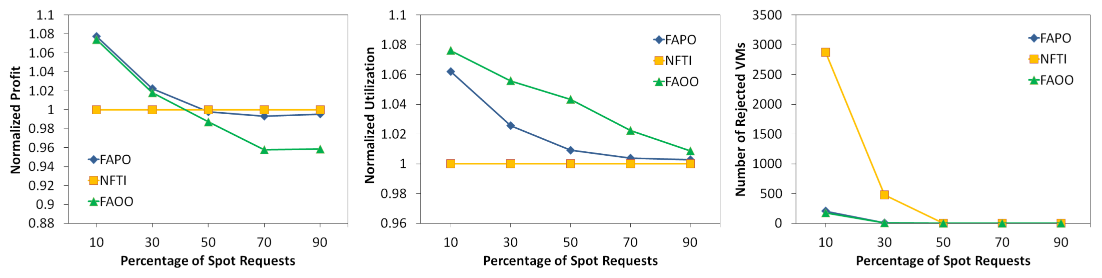
Results presented for profit and utilization are the normalized values for each metric using the result obtained form the *NFTI* policy as the base value. Since the *NFTI* policy reflects the situation where providers do not explore capacities of the federation, the use of normalized values allows us to quantify the benefits of federation-aware policies on each provider.

Impact of percentage of spot requests

The first experiment aims at evaluating how changes in the ratio between spot and on-demand requests affect performance metrics.

This experiment's simulation scenario consists of 5 providers, each one with a workload whose *aarr* value is 6.7 and 40% of the spot requests as persistent requests. The workload for each provider was generated as described in Section 3.5.2, but the percentage of spot VM requests (β) varied between 10% and 90% of the total amount of requests submitted to each provider. Results for this scenario are presented in Figure 3.2.

Figure 3.2a shows that, for smaller amount of spot VM requests, exploiting the potential of federation for outsourcing or contributing to the federation helps providers to enhance profit. After the point that 50% of requests are spot requests, providers are not able to increase profit since spot VM price is the most effective factor on the profit.



(a) (b) (c)

Figure 3.2: Impact of percentage of spot requests on (a) Profit (b) Utilization, and (c) Number of rejected on-demand VMs for a provider with different policies.

Moreover, when a significant number of running VMs in the data center belongs to spot requests, providers are able to absorb spikes in demand just by terminating spot VMs.

Besides the fact that the presence of more spot VMs in data center decreases the potential of the federation for making profit, the *FAPO* policy has better performance comparing to the *FAOO* policy with higher profit and less utilization.

As shown in Figure 3.2b, the utilization achieved with the *FAOO* and *FAPO* policies are always higher than with *NFTI*, because providers dedicate part of their capacity to the federation. However, *FAPO* witnesses smaller utilization, although it generates more profit than *FAOO*. This is due to the fact that *FAOO* contributes to the federation more than *FAPO*, and also terminates less spot VMs than *FAPO*, which results in lower spot price.

As we expected, by increasing the amount of spot requests, it is less likely to reject on-demand requests, because the chance of finding a spot VM to be terminated increases. Moreover, Figure 3.2c shows that less rejections occur for those policies that benefit from federation, especially when the percentage of spot VMs is low.

Impact of percentage of persistent spot requests

The second experiment demonstrates the effects of changing the percentage of persistent spot VMs requests on providers. This experiment's simulation scenario consists of 5 data centers, with *aarr* value of 6.7 and 30% of spot requests among the total number of requests for each provider. The workload for each provider was generated as described in

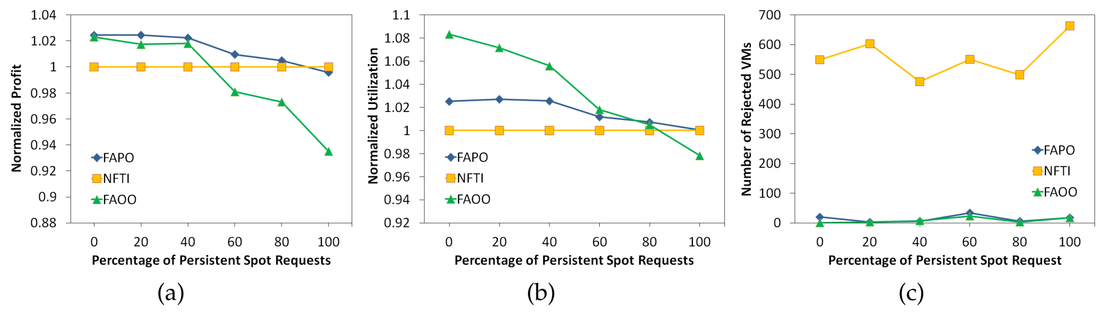


Figure 3.3: Impact of percentage of persistent spot requests on (a) Profit (b) Utilization, and (c) Number of rejected on-demand VMs, for a provider with policies.

Section 3.5.2, but the percentage of persistent spot VMs (α) varied between 0% and 100% of the total amount of spot requests submitted to each provider. Results for this scenario are presented in Figure 3.3.

More persistent spot VM requests results in less usage discontinuation, since even after termination of spot VMs the system retains the requests themselves, which can be served in a later stage. Percentage of persistent spot VMs is significant, since termination of spots VMs that are persistent does not cause load and revenue loss, but increases the current spot price, and consequently provider profit.

Therefore, according to Figure 3.3a, profit making of the *FAOO* policy drastically decreases after a point where 50% of spot market is used by persistent spot VMs, because this policy causes less spot VMs termination than other policies. The *FAPO* policy shows better performance in comparison to other policies regarding profit, as it benefits from outsourcing in lower percentage of persistent spot VMs, and termination of spot VMs in higher percentage of persistent spot VMs.

It is expected that a smooth increase in utilization will occur when there are more persistent spot VMs, however, it is not observable in Figure 3.3b. This is because spot termination does not result in losing part of VM requests. Thus, when there is a higher percentage of persistent spot VMs, policies converge to a specific utilization point because the total load remains constant in higher persistency of spot VMs.

Finally, the percentage of persistent spot requests does not have a significant effect on the number of rejected on-demand VMs. However, due to the lack of outsourcing choice,

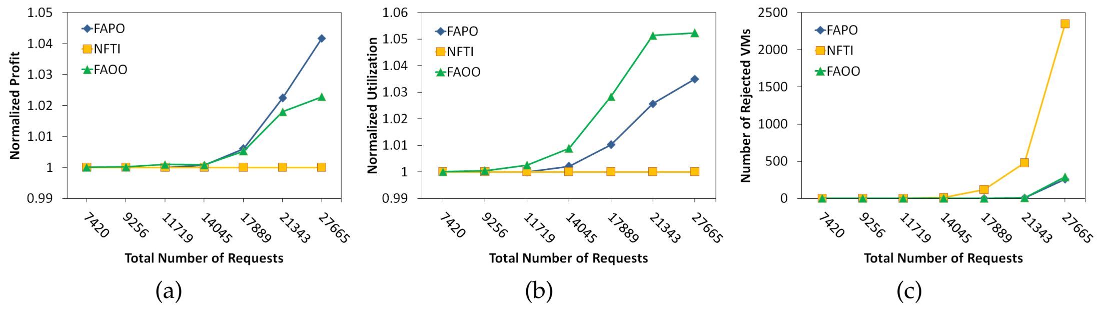


Figure 3.4: Impact of load on (a) Profit (b) Utilization, and (c) Number of rejected on-demand VMs, for a provider with different policies.

a higher number of rejected requests is seen when *NFTI* is applied.

Impact of the load

The third experiment evaluates the effect of load variation. The scenario consists of 5 data centers, α of 40% and β of 30%. The workload for each provider was generated as described in Section 3.5.2, but the *aarr* parameter of the Lublin workload varied between 8.2 and 6.4 to vary the number of generated requests for all the providers.

Figure 3.4 shows the impact of varying the *aarr* parameter, which results in a different number of requests, on the proposed policies. Since our policies are triggered when the provider is fully utilized, load is the most influential parameter in our experiments. By increasing the number of total requests, and consequently provider's load, the provider frequently experiences a situation where it has to decide between outsourcing and terminating spot VMs. According to Figures 3.4a and 3.4b, *FAPO* and *FAOO*, which support outsourcing, have higher profit and utilization by increasing load. However, the *FAPO* policy outperforms the *FAOO* policy by having a higher profit with smaller utilization. The difference between *FAPO* and *FAOO* is significant at higher loads.

By increasing the number of requests, the number of on-demand VM rejection also increases, because providers are subject to a higher load. The *NFTI* policy is more sensitive to this effect, because it does not support the outsourcing option.

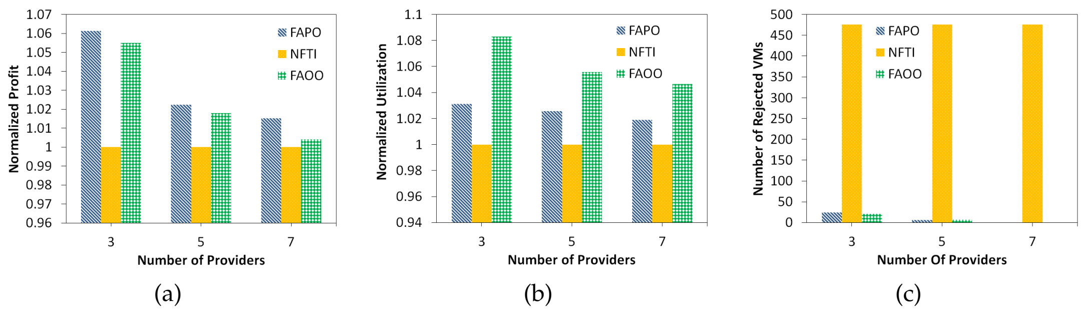


Figure 3.5: Impact of number of providers on (a) Profit (b) Utilization, and (c) Number of rejected on-demand VMs for a provider with different policies.

Impact of number of providers in the federation

This experiment evaluates the impact of number of participants in the federation on the results delivered by each policy. In this experiment, α is 40%, β is 30%, and $aarr$ is 6.7. The experiment was repeated with 3, 5, and 7 providers. The results are presented in Figure 3.5.

By increasing the number of providers, policies with an outsourcing option have a smaller number of rejected on-demand VMs, because it is more likely that the provider can find another provider that can serve an outsourcing request. Increasing the number of providers does not have any impact in *NFTI*, as in this policy there is no interaction with federation members. For *FAOO* and *FAPO*, an increase in the number of providers in the federation results in lower profit, because of better matching in supply and demand. That is, the offer price for contributing to the federation falls down and outsourcing becomes more profitable.

3.6 Summary and Conclusion

We proposed the policies to enhance IaaS providers' profit when the provider is a member of a cloud federation. Since each provider has the restricted amount of capacity, increase in load may overload the provider's data center and may result in QoS violation or users' request rejection. Providers that support different types of QoS and pricing scheme for VMs (e.g. on-demand and spot VMs) have the possibility of canceling their terminable less profitable VMs (e.g. spot VMs) in favor of more profitable requests (e.g. on-demand VMs). Providers can also benefit from federation by outsourcing requests to other members of the federation with smaller load.

Various experiments conducted to determine the impact of a provider's decision on its performance metrics. Evaluated parameters include the ratio of spot VMs to the total load, percentage of persistent spot VMs, number of providers in the federation and provider's load. Results showed that our policies help providers to enhance profit and to decrease the number of request rejections, while they keep utilization at an acceptable level.

Experimental results also allow us to derive some guidelines for providers. Running on-demand requests locally is more profitable when the provider has a high ratio of spot VMs, and the termination of spot VMs leads to less discontinuation of service usage by customers (i.e., high number of persistent spot requests). Moreover, outsourcing is more profitable when spot VMs are scarce and spot VM termination result in discontinuation service usage by customers. Furthermore, federation also helps underutilized providers in making more profit by selling idle resources to other members.

Chapter 4
Financial Option Market Model

Pay-per-use service by cloud service providers has attracted customers in the recent past and is still evolving. Since the resources being dealt within clouds are non-storable and the physical resources need to be replaced very often, pricing the service in a way that would return profit on the initial capital investments to the service providers has been a major issue. Moreover, to maintain Quality of Service (QoS) to customers who reserve the resources in advance and may or may not be using the resources at a future date makes the resources wasted, if not allocated to other on-demand users. Therefore, a need for a mechanism to guarantee the resources to reserved users whenever they need them, while keeping the resources busy all the time is in very high demand. The concept of federation of cloud service providers has been proposed in the past wherein resources are traded between the providers whenever need arises. We propose a financial option based cloud resources pricing model to address the above situation. This model allows a provider to hedge the critical and risky situation of reserved users requesting the resources while all the resources have been allocated to other users, by trading (buying or outsourcing) resources from other service providers in the cloud federation. We show that using financial option based contracts between cloud providers in a cloud federation, providers are able to enhance profit and acquire the needed resources at any given time. It would also help creating a trust and goodwill from the clients on the cloud service providers with less number of Service Level Agreement (SLA) violation.

4.1 Introduction

CLOUD providers usually offer customers two well-known payment plans: *reservation* and *on-demand*. Amazon EC2[1] and GoGrid[2], for example, provide reservation and on-demand plans of the infrastructure services. Customers pay in advance to reserve

[1]Amazon EC2, `http://www.aws.amazon.con/ec2/`.
[2]GoGrid, `http://www.gogrid.com/`.

the instances for the possible future usage and in exchange receive a significant discount on the charge for running VMs in the reserved capacity. Moreover, customers receive higher availability of the service for reserved than on-demand instances. On the one hand, reservation plans allow the customers to acquire resources in cheaper price and higher availability than that of on-demand plans. On the other hand, it helps providers to attain more efficient resource management and procurement. In addition, reservation can guarantee cash flow even if the reserved resources are not fully utilized by the customers.

Since cloud applications such as web applications experience huge and unpredictable variation in the load over time, defining the required amount of instances to cope with the load experienced in a given moment is a challenging task for the users. If the load was known beforehand, users could reserve the required amount of instances, which is cheaper than acquiring on-demand instances. However, as loads are unpredictable and variable, users have to combine reserved instances with on-demand instances for the situations in which the former is not enough [20]. This provides a balance between cost and utilization of the resources.

The pattern of utilization at user side causes reserved instances not to be deployed at all times. This offers providers the opportunity to explore this underutilized reserved capacity for additional cash flow by releasing them to the on-demand requests. Therefore, if the unreserved part of the data center experiences high utilization, providers are able to accommodate on-demand requests on the underutilized reserved capacity of the data center. However, providers are liable to provide guaranteed availability for the reserved requests according to the service level agreement (SLA). Consequently, providers face a risk of SLA violation by using the reserved capacity for accommodating on-demand requests.

Cloud cooperation is a possible solution in order to hedge against the mentioned risks by letting providers increase their resources dynamically [14, 99]. Recent works demonstrate that federation of providers and interoperability between clouds to trade resources in a market helps providers to enhance their profit, resource utilization, and QoS [35]. The use of shared pool of physical nodes for on-demand and reserved instances along with outsourcing requests substantially mitigate the risk of SLA violation for reserved instances. But, the provider still faces the other risk of being unable to acquire required resources in the federation market. Essentially, they may end up short selling resources without having a good knowledge of usage loads and hence violating the promised QoS. Furthermore, according to the efficient market hypothesis in economic markets, providers can not precisely predict the future price variations in the federation market using past price history [96, 109].

In this chapter, a *financial option-based* market model is introduced for a federation of cloud providers, which helps providers increase their profit and mitigate the risks (risk of violating SLA and risk of paying extra money). A financial option [46] is a contract for a future transaction between two parties: *holder* and *seller* of the contract. A financial option gives the holder the right, but not the obligation, to buy (or to sell) an underlying asset at a certain price, called the *strike price* (exercise price), within a certain period of time, called *maturity date* (expiration date). The seller is obligated to fulfill the transaction. As a compensation the seller collects an upfront payment at the beginning of the contract, called *premium*. In our proposed framework model, a provider buys option contracts as a backup capacity for the reserved resources used by on-demand instances, to gain the right to acquire resource from the seller provider, as need arises. Since the seller is obligated to fulfill the request, risk of not acquiring resources is removed. Moreover, buying option contracts protects provider against high variation of the market price. In summary, this chapter has the following main contributions:

1. A financial option-based market model in the presence of the cloud federation to help providers to manage their reserved capacity and achieve higher QoS guarantee.

2. Evaluation of the proposed model to show its effectiveness in increasing provider's profit without imposing any SLA violation.

The remainder part of the chapter is organized as follows: In the next section we give an overview of the related work. The system model and problem definition are identified in Section 3.3. Our proposed option model including the parameters setting and pricing mechanism is explored in Section 4.4. In Section 4.5, the baseline policies and the proposed policy based on our model is introduced. A detailed discussion on workload and simulation setup, performance metrics, and experimental results are given in Section 4.6. Finally, Section 4.7 presents the conclusions.

4.2 Related Work

Resource provisioning for IaaS cloud providers is a challenging issue because of the high variability in the load over the time. Providers must be able to dynamically increase the available resources to serve requests [35]. In order to enable such scenario, coordination between providers has to be achieved, possibly through the establishment of a cloud federation. In recent years, different platforms for cloud federation have been proposed in the literature [14, 99, 101]. Economic aspects of the cloud federation including motivations and incentives for parties joining the federation have been investigated by several studies [35].

Works related to systems for market-making such as works by Song et al. [112], Mihailescu and Teo [71], Gomes et al. [40], and Vanmechelen et al. [119] concern about mechanisms for creating markets and trading resources. In this chapter, a financial option-based market model has been introduced for a federation of cloud providers. An introduction to the foundations and basics of financial option theory can be found in [46].

The authors in [2] propose a model based on financial option theory to price Grid resources. They use option for pricing Grid resources in order to maintain equilibrium between service satisfaction of Grid users and profitability of the service providers. They do not propose a model to sell and buy options as we do and their model proposed for a single resource provider environment. In our study, we consider multiple providers in a cloud federation where option contracts are traded between service providers. Moreover, note that the risk factors they are concerned with are different from the risks investigated in this chapter.

Another work devoted to option theory in resource allocation for clouds, proposes an approach based on the option theory to minimize cost and mitigate the risk for cloud users [96]. They introduce a novel pricing scheme based on the option that cloud providers should provide for their own customers. Using option plan, customers can reduce the cost of using IaaS cloud provider resources. Our work, on the other hand, mainly aims to increase profit and mitigate risks for providers, which leads to better QoS for the customers.

Meinl and Neumann [65] analyze the use of real options in a contract market, to economically manage resource reservation in distributed IT environments. In fact, they use option as a contract to perform reservation for time and budget sensitive customers. Grid consumers want to minimize expenses, whereas Grid providers want to maximize their return on investment. Our work is similar as we also focus on reservation, but we use option as a hedging mechanism for the reserved capacity to enhance provider profit. Besides, our reservation scheme is also different. They consider reservation for Grid jobs with deadline and budget while our reservation scheme is like what IaaS cloud providers offer.

Bossenbroek et al. [13] investigate the application of option contracts in the context of a market for Grid resources, in order to deal with price volatility. They assess the performance of three classic hedging strategies for buying options in this regard. In order to define the underlying asset, the concept of *leases* is introduced. These correspond to a right to use a Grid resource for a fixed time period (e.g. one hour). The size of a task is expressed in a multiple of such leases, and it is assumed that the task's load is entirely divisible over such leases. The model is therefore not directly applicable to the cloud computing domain considered in this chapter.

Cloud providers offer customers reservation (e.g., prepaid) and on-demand plan (e.g., pay per use). There are always incentives for both cloud providers and customers to use reservation. For example, reservation may result in better capacity planning, guaranteed cash flow for providers, and availability of resources and discount on the usage of such resources for customers. However, it is important for the customers to optimize the amount of resources that they reserve to reduce the cost. Reservation is a challenging issue, since it should be done in advance when there is an uncertainty about the actual future demand. Under-provisioning (Reserving less) and over-provisioning (Reserving

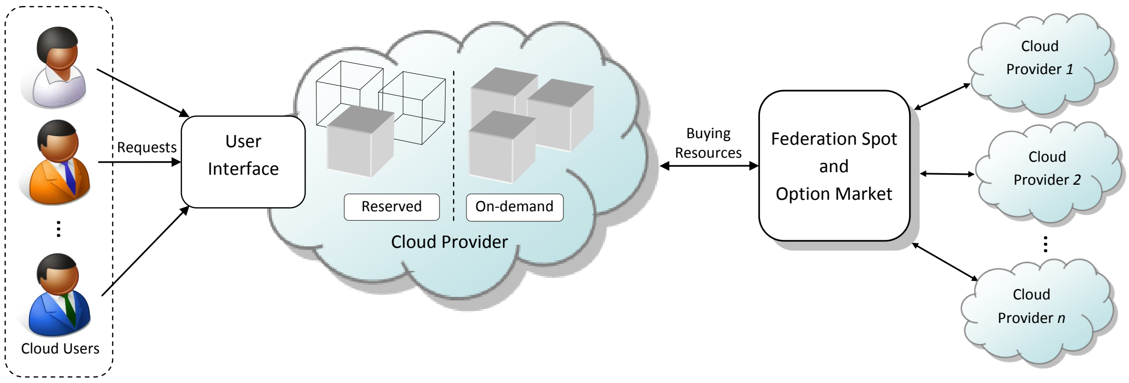


Figure 4.1: Model elements and architecture.

more) of reserved instances may result in extra costs for the customers. Authors in [20] propose an algorithm to minimize total cost of resource provisioning and avoid over-provisioning and under-provisioning of reserved instances.

Cloud providers supporting reservation should answer the question of how to allocate resources between reserved and other types of requests to maximize their revenue. A sample solution for this can be found in [63].

4.3 The System Model

In this section, we outline the system model including the markets, market participants, their motivations, and all the corresponding parameters of the model (Figure 4.1).

Cloud users (IaaS cloud customers) submit their IT infrastructure requirements in the form of VM requests to the *Cloud provider* (see Figure 4.1). Such a request submission may lead to acquisition of VM instances for a certain amount of time by the customer. Since customers decide when to terminate the instances, the cloud provider does not have a priori knowledge of the holding time of the instances (lifetime of instances). Requests can be submitted either for *reserved* or *on-demand* service and charges will be applied accordingly.

1. *On-demand plan*: On-demand plan allows customers to pay for compute capacity by its usage without long-term commitment. Price is calculated at a fixed rate per usage time, e.g., hourly, from the time an instance is launched until it is terminated.

If the provider possesses enough resources, resource provisioning for VM requests is done, otherwise the request is rejected by the provider. After instantiation of VMs, customers can retain machines as long as they require them.

2. *Reserved plan*: In this plan, customers pay an upfront fee, called reservation fee, and in return receive a discount on the usage for the VMs. Reserved plan also assures that the reserved capacity is always available when it is required.

We believe that customers do not always fully utilize the reserved capacity in the reservation lifetime. Partial utilization of the reserved capacity still has benefit for customers. For example Amazon EC2 users gain economic advantage of using *Reserved* Instances in comparison with *On-Demand* Instances, even if they can utilize only slightly more than 8% of the reserved capacity in a 3-year contract[3].

The *Cloud provider* offers its resources for each plan based on the fixed price. The cloud provider offers best-effort and highly available service for on-demand and reserved instances respectively. Allocating data center capacity (physical nodes) to reserved and on-demand instances in order to meet the QoS of each plan is performed by the provider. Suppose that the provider has a data center with a predefined capacity and it is able to accommodate maximum n VM instances of similar types simultaneously. Two different strategies can be assumed to structure such a system to support aforementioned plans [63]:

1. *Isolated pools*: In this strategy, two different pools of servers (nodes in data center) for instances of each plan is considered in isolation of each other. The number of nodes for reserved instances, in this case, is defined according to the total number of reserved VMs by customers. If the entire reserved capacity size is r instances, the on-demand pool is capable of accommodating $n - r$ instances at most.

2. *Shared pool*: In the shared pool strategy, on-demand requests are offered to use physical nodes of the reserved capacity if on-demand capacity is fully utilized by on-demand VMs. In the shared pool strategy, If the data center maximum capacity in

[3]http://aws.amazon.com/ec2/reserved-instances/

unit of VMs of the similar type is n, the reserved capacity size is $r(r \leq n)$, and m reserved VMs are running $(m \leq r)$ then accommodating $n - m$ on-demand requests is possible while m remains unchanged.

Given that reserved instances and on-demand instances are just different in pricing and they function identically during execution, there is no technical barrier to set up the shared pool strategy. Note that the shared pool strategy suffers from the risk of violating availability of the reserved instances.

A *Cloud federation* allows providers to trade their resources through Federation Level Agreements (FLA) (see Chapter 3). In this paradigm, providers aim to overcome the resource limitation by buying resources from the market. Underutilized providers sell their resources in this market usually at cheaper prices compared to what they would charge their own customer, in order to avoid wasting their non-storable compute resources (e.g., *SpotCloud*[4]). It is worth mentioning that our model can also be applied to a hybrid cloud scenario in which a private cloud provider buys option contracts from a large public cloud provider.

The main element in our model is the *federation spot market* in which a group of federated clouds trade their resources with each other (see Figure 4.1). Note that, here the spot market is an exchange market in which on-demand resources are traded for immediate delivery similar to the market model presented in Chapter 3. Federation spot market should not be mixed up with the local spot market and spot instances presented in the previous chapter. From now onwards by the spot market we mean federation exchange market for immediate delivery of on-demand resources unless otherwise mentioned.

Different types of underlying market mechanism can be considered for the federation spot market, such as combinatorial double auctions, commodity exchanges, reverse Dutch auctions, and etc. The main focus of the current work is to build an *option market* on top of the federation spot market. Due to general nature of our proposed model, the option market is modeled independently from the underlying spot market mechanism. The only outcome of the spot market which is required by the model is the spot price at which resources are offered. Therefore, in our setting we do not specify a particular

[4]SpotCloud, `http://www.spotcloud.com/`.

spot market mechanism, and the scheme does not directly influence the model. In this chapter, we assume that the IaaS cloud provider is able to buy resources from the spot market at the current spot price. Strategies regarding selling resources to the market will be explored as an extension of this work.

The cloud provider in our model attempts to increase its revenue by using a shared pool strategy. This avoids wasting the underutilized reserved capacity by serving excess on-demand requests on that capacity. However, the IaaS cloud provider faces the risk of violating availability for the reserved requests. Thus joining the federation spot market could mitigate the risk by allowing the provider to outsource reserved and on-demand requests. However, a cloud provider participating in the federation spot market could still bear two risks, namely:

1. the risk of price fluctuation in the spot market and high cost of outsourcing, and
2. the risk of not being able to acquire resources, which leads to rejection of the reserved requests.

In order to hedge against above risks, we propose a market model based on the financial option on top of the spot market for a federation of cloud providers. Using our model, the provider is able to enhance its profit by deploying a shared pool of physical nodes for on-demand and reserved requests. Moreover, the provider ensures the availability of the reserved instances and avoids buying resources at a price that is higher than the one charged to its own customers.

In the current model, each time the provider accommodates an on-demand request in the reserved capacity due to lack of space in the on-demand pool, it buys an option to hedge against the situation of running short of resources for reserved requests. The option is exercised (i.e., the requests are outsourced to other cloud providers in the federation) if the reserved request arrives and the provider does not have enough resources to serve it locally. Reserved requests are submitted by customers based on the previously submitted reservation contracts.

The important advantage of buying options in comparison to other future agreements is that it gives the provider the right (not the obligation) to buy resources (outsource requests) in the future. Therefore, if the cloud client does not request the reserved instances, the provider will simply let the contract expire without responsibility to buy unnecessary resources. The only cost for providers in such an arrangement is the premium paid at the beginning of the contract. This cost, however, can translate into trust and goodwill by the clients on the provider.

In our model, providers transfer the risk of violating SLAs to other providers by buying option contracts and paying option premium. Therefore, sellers of the option contracts must consider the trade-off between the risk and expected profit. However, the scope of the current work is limited to buyer's strategies for purchasing required options. Strategies regarding selling options require further attention. Interested readers are referred to the work of Markowitz et al. on decisions under uncertainty in financial markets [62], and the work of Michalk et al. [68] on the translation of the model for cloud Service providers.

4.4 The Option Market

This section provides the detailed discussion of the option market including pricing, buying mechanism and exercising options. The main contribution of this chapter is to propose a financial option-based market mechanism for a cloud federation. The main element in such a market is the *option*. There are two types of option: *call* and *put*. A call option gives its holder the right to buy the underlying asset at a specific price (strike price) by a certain time (expiration date) [46]. A put option gives the holder the right to sell the asset at a specific price over a given period of time. Providers with large amounts of physical resources may buy put options that will give them the right to sell resources at their will. However, we do not consider the use of put option in this work.

Suppose that a market participant purchases a call option for \$2 with strike price of \$28 for expiration date in two months. Within two months, the spot price goes to \$35, in this case, he/she exercises the option and gains the advantage of $(35 - 28) - 2 =$ \$5. If the spot price stays below the strike price, the option holder might buy at the spot price and allow the option to be expired. In this case, the \$2 premium paid at the beginning of the option contract is lost.

An option contract is defined by a tuple (P, K, T), where P is the price of buying the option, K is the strike price, and T is the expiration date. In our model, the provider buys an option, whenever it accommodates an on-demand request in the reserved capacity. Terms of the option contract (P, K and T) need to be determined at the time the contract is signed. In the following paragraphs, the way we set the terms of the contract is explained.

Given that the price per time unit for the reserved instances is R, the provider buys an option with a strike price lower than R to secure its future profitability. It means that the provider assures that the price it pays to outsource a reserved request is always lower or equal to the price it charges its own customers, i.e., R. As long as the spot market price, S, is lower than R, the provider submits a request for buying an option with strike price $K = S$, otherwise $K = R$.

The provider is oblivious to the duration of the VM and its future load, so we consider T as a fixed value, e.g., one month. We investigate the impact of the time to maturity of the options on the model in Section 4.6. Optimization strategies regarding buying option with the best expiration date requires load prediction strategies. It can be considered as an extension of this work.

The option can either be exercised at expiration date (*European option*), or any time during its life (*American option*). Since the provider buys an option to hedge the risks for reserved requests and needs to exercise the option any time in the future according to the load and upcoming reserved requests, the American call option is the most appropriate for our work.

When the provider desires to purchase an option, it needs to pay the option price or the premium to the seller. The value of an option (option price or premium) can be estimated using a variety of quantitative techniques based on the concept of risk neutral pricing [12, 67].

A useful and popular technique for pricing an option involves constructing the price movement in a structured manner known as binomial lattice or tree [23]. The binomial tree represents different possible paths that might be followed by the underlying asset price over the life of the option. In our model, the underlying asset is the available re-

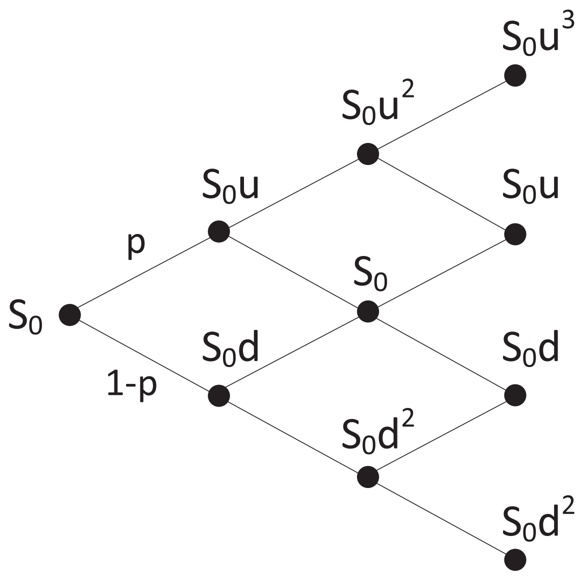


Figure 4.2: Binomial tree for option pricing.

source in the federation market.

Consider the current spot market price is S_0. S_0 goes to S_0u with probability of p and to S_0d with probability $1-p$ at each time step ΔT. Let $T = n.\Delta T$, where T is the option expiration date, then a lattice of spot price movement for $n = 3$ is presented in Figure 4.2. The value of the option can be evaluated for each point at the leaf nodes of tree (time T). The value of the option at starting node can be calculated through a procedure known as backward induction. A call option is worth $max(S_T - K, 0)$, where S_T is the spot market price for underlying asset at time T.

Assuming risk neutral world, the value at each node at time $T - \Delta T$ is computed according to the expected pay-off value at time T and the risk-free interest rate, r, for the time period ΔT. In this study we assumed $r = 0$. Going backward using the above procedure, the option value, P, can be obtained at time zero. American call option that pays no dividend is never exercised early. Consequently, the following procedure is valid for both American and European call options [46].

Factors u, d and p play crucial role in option pricing. All of the aforementioned parameters can be calculated according to a parameter called *volatility*. The volatility, σ, in stock market is a measure of uncertainty about the returns provided by the stock and future stock price. There is a wealth of literature on calculating σ in finance community [95]. One common method to obtain the volatility is by using the history of the stock price movements. We used the method provided in [46] to estimate volatility according to the historical data. Choosing a proper size for the time frame of the historical data to calculate σ is non-trivial. In this chapter, we set this value equal to the maturity time for the option contract. u, d, and p can be calculated according to σ as:

$$p = \frac{1-d}{u-d}, \quad u = e^{\sigma\sqrt{\Delta T}}, \quad d = e^{-\sigma\sqrt{\Delta T}}$$

4.5 Policies

Three different policies to evaluate our model including two baseline policies and a policy using our option market model are described in this section. We first present baseline polices based on an isolated pool of servers with and without federation respectively, and then the federation option-market enabled policy using a shared pool of physical nodes is proposed.

4.5.1 Baseline In-house Isolated Pool Policy (IIP)

The first baseline policy is the simplest policy in which the provider works independently, without participating in the federation. Moreover, the provider in this policy uses isolated pools of physical nodes for on-demand and reserved instances in order to guarantee high availability of reserved requests. As a result this policy rejects extra on-demand requests even when the reserved capacity of the data center is not fully utilized.

4.5.2 Baseline Federated Isolated Pool Policy (FIP)

In this policy, we assume that the provider is able to access the federation spot market. Resources are bought at the spot price from the cloud federation market to outsource on-demand requests if the provider is not able to serve them locally. To be always cost-effective, if the spot price in the federation market is higher than the local on-demand price then the provider rejects the on-demand request. In this policy, the pool of physical nodes for reserved and on-demand instances are isolated to prevent rejection of reserved requests.

4.5.3 Federated Shared Pool Option-Enabled Policy (FSPO)

The third policy exploit the potential of our proposed option market model to hedge against risk of using the shared pool strategy. In this policy, the provider accommodates excess on-demand requests in the underutilized reserved capacity. To facilitate this, the provider buys an option whenever it accommodates on-demand requests in the reserved capacity. Consequently, if a reserved request comes in and the provider is not able to serve it locally, the option is exercised and the reserved request is outsourced at the strike price of the option.

It is necessary to mention that policies with the shared pool strategy without considering our proposed option model (e.g., In-house Shared Pool or Federated Shared Pool) are not taken into account here. Those policies might lead to rejection of reserved requests and QoS violation. However, the number of rejections occurring for the reserved requests for those policies are reported in our experimental evaluation.

4.6 Performance Evaluation

4.6.1 Experimental Setup

Workload setup

Due to the lack of publicly available traces of real-world IaaS cloud requests, we created a synthetic workload model to generate VM requests for an IaaS cloud.

Our workload model complies with previously reported workloads for IaaS providers in the literature [35,74]. Generating a VM request requires a pair (S, D), where S is the arrival time of the request and D is the holding time of the instance by the customer.

In order to model the holding time of the instance by users, D is taken to be a Pareto

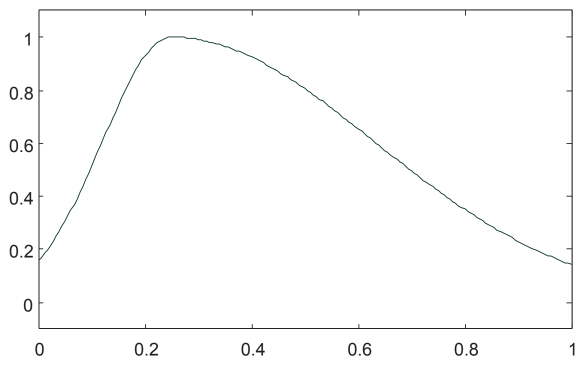


Figure 4.3: Combination of two Gaussian functions.

distributed random variable, with shape parameter $\alpha = 1.1$ and location parameter $\beta = 1$ [74]. The Probability Density Function (PDF) of the Pareto distribution is

$$f(x) = \begin{cases} \alpha\beta^{\alpha}/x^{\alpha+1} & \text{for } x \geq \beta \\ 0 & \text{for } x < \beta \end{cases}. \tag{4.1}$$

The value of the random variable D represents the holding time of the VM by the user in the scale of hours.

Arrival times of the requests are generated as follows. Given that our workload follows daily pattern, the combination of two Gaussian functions in a range of $[0, 1]$ with the given equation has been chosen. A Gaussian function is defined by Equation 4.2.

$$f(x) = e^{-\frac{(x-a)^2}{2b^2}} \tag{4.2}$$

The function is shown in Figure 4.3. As it can be seen in Figure 4.3, the function can be fit into two-parts for two sets of values of a and b. With $a = 0.13$ and $b = 0.25$ the first part of the curve can be fit; and with $a = 0.38$ and $b = 0.13$, the second part of the curve is fit. The resulting shape in Figure 4.3 has been divided in 24 buckets of equal width, where buckets are related to a specific hour in a day and they starts at 4:00 AM. Then, the proportion of the number of requests arriving every hour of the day is calculated according to the ratio of the area in a bucket to the total area under the curve. Afterwards, we generate a uniformly distributed arrival time for requests in every hour. In order to create the weekly patterns for the workload, we divided days per week in two

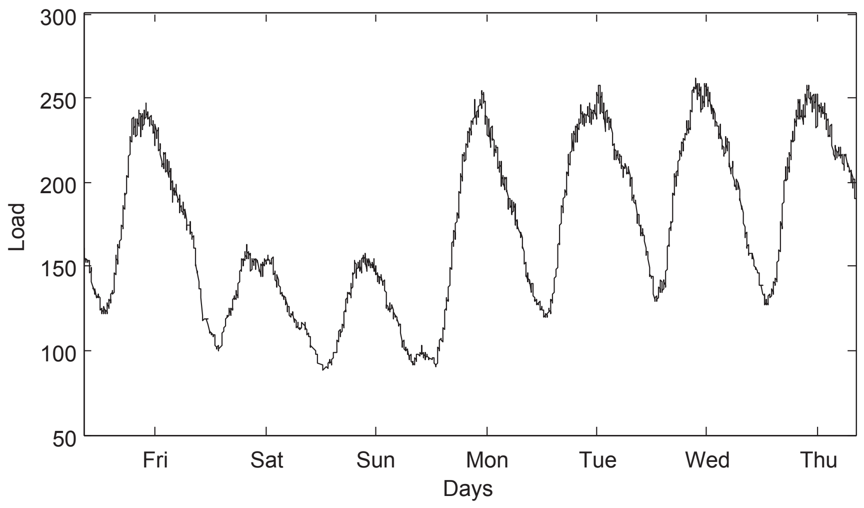


Figure 4.4: Generated workload during a week.

parts: weekdays and weekend (Saturday and Sunday) with a lower number of requests for the weekend (reduction of 50%). In our simulation the number of requests per day in weekdays and weekends vary to increase or decrease the data center load. An example of load generated via our workload model for one week is depicted in Figure 4.4.

We intend to study the behavior of the financial option-based market model in different situations of the load. For this purpose, our workload model is applied to generate reserved and on-demand requests separately with different values for weekdays and weekends.

Simulation Setup

The experiments presented in this work were developed using the CloudSim [16], a discrete-event cloud simulator. The simulated scenario is created according to the model in Section 4.3.

Considering the components in Figure 4.1, the cloud provider in our simulation receives the VM requests generated by the described workload model in the previous subsection. Requests are either on-demand or reserved. The VM configuration is inspired by Amazon Elastic Compute cloud (Amazon EC2) small instances.[5] The pricing is also adopted from the Amazon EC2 price of small instances in the US east region for on-

[5]Small instance: 1 CPU core, 1.7 GB RAM, 1 EC2 Compute Unit, and 160 GB of local storage

demand and medium utilization reserved instances at the time of the experiment.[6] The provider charges their customers based on hourly usage, at the cost of \$0.085 and \$0.030 per hour for on-demand and reserved VMs, respectively. We do not take into account the premium fee for the reservation contract as it only adds a constant value to the provider revenue.

The provider capacity is set equal to the maximum number of simultaneously runnable VMs. In our simulation, the provider capacity is set to 200 VMs. For the sake of better analysis, the reserved capacity of the data center is considered as steady constant value for the whole simulation. The reserved capacity is set to 100 VMs in all the experiments.

Federation spot market prices are generated according to the statistical model presented in [47]. The model fits well with the price fluctuation of the spot instances in the Amazon EC2 spot market. The model is applied in our work because of two main reasons. First, correlation between generated spot prices and the price of on-demand and reserved instances makes the evaluation of our model more significant. Second, the statistical model provides more flexibility to investigate our proposed market mechanism in comparison to the spot price traces of Amazon as we can modify the parameters.

Option pricing is done by the previously described method in Section 4.4. The depth of the binomial tree to calculate the option value is 30. Volatility, as we mentioned earlier, is calculated according to the examination of the past spot prices leading to the contract time with the same length of option maturity time.

In our experiments, the length of a simulation period is 6 months. Each experiment is carried out 30 times and the mean value of the results are reported.

4.6.2 Performance Metrics

Two different performance metrics for evaluation purposes have been used in our experiments. Our model has been developed to increase the provider's profit while availability of the reserved instance remains high. Therefore, profit has been selected as a first measure to evaluate the model. Provider's profit calculated according to the following

[6] Amazon EC2 pricing, `http://aws.amazon.com/ec2/pricing/`.

equation:

$$P = R_O + R_R - C_{out,O} - C_{out,R} - C_{option} - C_p \ , \tag{4.3}$$

where R_O and R_R are the revenue of the on-demand and reserved instances. $C_{out,O}$ and $C_{out,R}$ are cost of buying resources from another provider to outsource on-demand and reserved instances, respectively. All R_O, R_R, $C_{out,O}$ and $C_{out,R}$ are computed by the summation of the running expenses for the served VMs of each pricing plan during the experiment's period. The running expense of a VM is calculated according to the price per hour multiplied by the instance-hours consumed for each VM, from the time an instance is started until it is terminated. Each partial instance-hour is considered as a full hour. The reservation fee for the reserved instances is not considered here as it imposes a same fixed value in calculation of the R_R for all the policies. C_{option} is the total cost for buying options in the federation (premium costs) incurred by the provider. Finally, C_P is the provider's cost, which includes the operational cost of the data center (i.e., hardware and software acquisition, staff salary, power consumption, cooling costs, physical space, amortization of facilities, etc.). We ignore C_p in our experiments as it imposes a constant value to all policies.

The second metric is the number of rejected reserved requests, which shows the number of SLA violations for the reserved instances.

4.6.3 Experimental Results

In the first set of experiments we evaluate the profitability of the model by changing the loads of the on-demand and reserved requests. First, all parameters of the system were fixed and the load of the reserved capacity was increased (Figure 4.5a). The capacity of the data center is 200 VMs, the reserved capacity is 100 VMs, the maturity time of the options is 30 days, and the number of on-demand requests per weekdays and weekends is 700 and 350, respectively.

Average utilization of the reserved capacity in the data center was changed by increasing the number of reserved requests. As shown in Figure 4.5a, the generated profit for

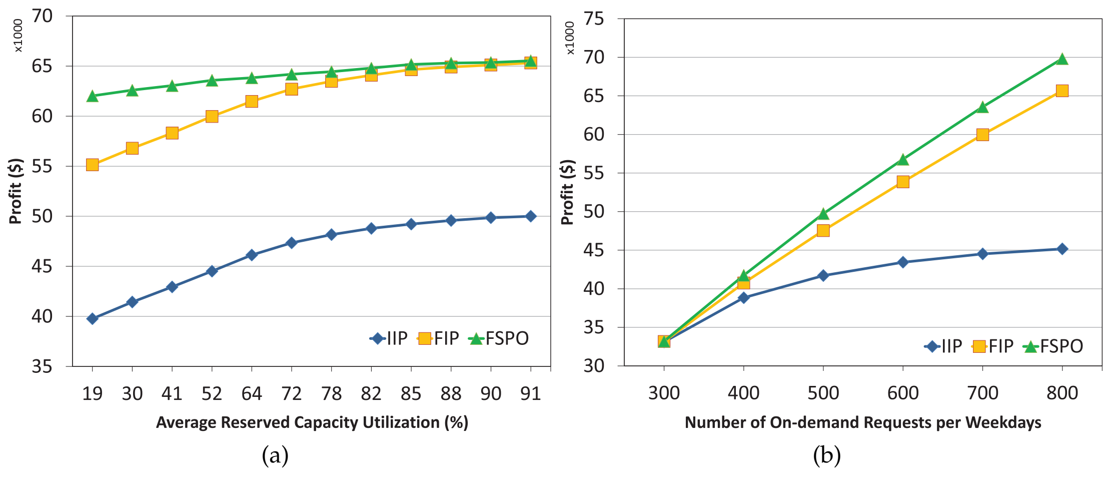


Figure 4.5: Impact of (a) the reserved capacity utilization and (b) the number of on-demand requests on provider's profit with different policies.

the IIP and FIP, which do not use the underutilized capacity of the reservation for accommodating on-demand VMs, rises smoothly with the increment of the reserved capacity utilization. The difference between IIP and FIP is caused by the outsourcing of the excess on-demand requests in the FIP. On the other hand, as the reserved capacity utilization is increased, FSPO experiences less growth in profit in comparison to the other policies as the underutilized capacity for in-house accommodation of the on-demand requests decreases. Eventually, FSPO generates the same profit as the FIP policy.

We use the same configuration of the previous experiment while utilization of the reserved capacity is fixed at 52% and the number of on-demand requests is varied from 300 to 800 per day for weekdays and half of that for weekends. All policies generate the same profit at low number of on-demand requests because outsourcing on-demand requests or accommodating them in the reserved capacity is not required. The IIP policy that does not benefit from the potential of outsourcing, generates less profit in comparison to the other policies. The FSPO is the most dominant policy as it utilizes both federation and the underutilized reserved capacity.

The objective of the third experiment is to examine the effects that the volatility of market prices has on the profit making opportunity from the proposed model. Since the prices generated by the spot pricing model proposed by Javadi et al. [47] to model

Table 4.1: Number of On-demand (O), Reserved (R), Rejected On-demand (RO), Rejected Reserved (RR), Outsourced On-demand (OO), and Outsourced Reserved(OR) requests for the provider with policies.

Policy	**O**	**R**	**RO**	**RR**	**OO**	**OR**
IIP	106340	36590	42450	**0**	0	0
In-house + Shared Pool	106340	36590	19188	**6636**	0	0
FIP	106340	36590	219	**0**	42230	0
Federated + Shared Pool	106340	36590	111	**38**	19078	6598
FSPO	106340	36590	111	**0**	19078	6636

Amazon EC2 spot prices has substantially low price volatility, we increase the standard deviations of the mixture of Gaussian distributions used by the model to achieve higher volatility. In order to remove unrealistic very low market prices, prices below \$0.025 were ignored. Simulation parameters for this experiment were set as follows. The capacity of the data center is 200 VMs, the reserved capacity is 100 VMs, the maturity time of the options is 30 days, and the number of on-demand requests per weekdays and weekends is 700 and 350, respectively, and the utilization of the reserved capacity is 64%. As shown in Figure 4.6, FSPO is more resilient to the higher degree of volatility in comparison to FIP, as it secures the outsourcing cost at prices below the price of reserved instances. Note that when the spot price fluctuates more, the provider has to buy options at a higher price to hedge against price variation.

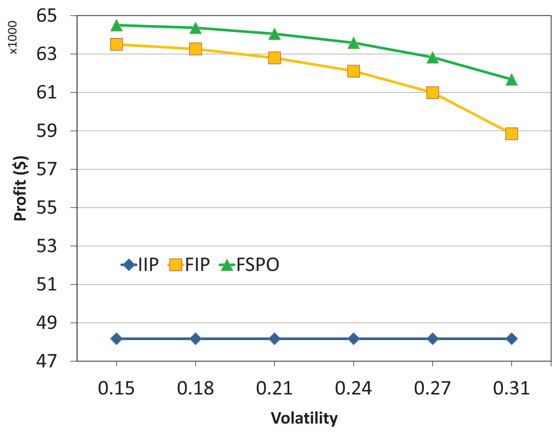


Figure 4.6: Impact of price volatility on behaviors of policies.

We added a shared pool strategy to the baseline policies. The number of rejected reserved instances is reported in Table 4.1. It demonstrates how the option model helps providers to hedge against the risk of SLA violations for the reserved instances. We considered that resources are not available in the spot market of the federation, if the spot price is higher than \$0.085 (on-demand price). The higher risk of unavailability in the market will cause more SLA violations for the *Federated Shared Pool* policy. The Federated Shared Pool policy shows a small number of rejections in our experiments, as the model we used here to generate spot prices according to the Amazon EC2 spot market rarely generates prices higher than the on-demand price.

We also investigate the impact of the maturity time of the option in profitability of the FSPO policy. The maturity time for the option contract was varied from 7 to 90 days and the generated profit was reported in Table 4.2. The configuration of this experiment is the same as the first experiment when the utilization of the reserved capacity is 52%. The results show that the maturity time of the option contracts does not have a significant impact on the gained profit because when the maturity time rises, the number of bought option contracts falls.

Table 4.2: Impact of option maturity time on provider's profit using FSPO.

Maturity time (Day)	**Bought Option**	**Profit (\$)**
7	3086	63588
10	2612	63588
30	1913	63583
60	1745	63578
90	1701	63574

4.7 Summary and Conclusion

Cloud providers usually offer on-demand and reserved plans. Since the resources traded in clouds are non-storable and the physical resources need to be replaced very often, exploiting part of the reserved capacity that is not used by the reserved users has a great benefit for the providers. Nevertheless, agreed Quality of Service (QoS) with customers who reserve resources in advance should be satisfied. Therefore, a need for a mechanism to guarantee resources to reserved users whenever they need them, while keeping resources loaded all the time is of value.

We proposed a financial option model for a federation of cloud providers to address the above situation. This model allows a provider to hedge the critical and risky situation of reserved users requesting the resources while all the resources have been allocated to other users, by trading (buy or outsource) resources from other service providers in the cloud federation.

Experimental results demonstrated that financial option based contracts between cloud providers in a cloud federation, would help them to exploit the underutilized reserved capacity without any concern to acquire the needed resources at any given time. The provider's profit will be increased by using our model, while the number of rejections of the reserved requests is negligible. The model therefore contributes to obtaining a trust and goodwill from the provider's client base.

Part II

Profit Maximization for a Single Cloud Provider

Introduction to Part II

AS popularity of cloud computing is rapidly growing and many more cloud providers are emerging, cost efficiency and revenue maximization become two major concerns of cloud providers to remain competitive while making profit. In the first part of the thesis, we investigated the profit maximization problem in a federated cloud environments where cloud providers cooperate to increase the degree of multiplexing. In this part, we outline novel economics-inspired resource allocation mechanisms to tackle the profit maximization problem from the perspective of a cloud provider acting solely. In chapter 5, we propose admission control mechanisms tailored within a revenue management framework to maximize revenue where various pricing plans in multiple marketplaces are supported by the provider. In chapter 6, we propose an auction-based dynamic pricing mechanism suitable for selling the spare capacity of the data center. In the subsequent chapter, we present a realization of the proposed dynamic pricing mechanism within a pricing as a service framework.

This page intentionally left blank.

Chapter 5

Revenue Management with Optimal Capacity Control

Infrastructure-as-a-Service cloud providers offer diverse purchasing options and pricing models, namely on-demand, reservation, and spot market plans. This allows them to efficiently target a variety of customer groups with distinct preferences and to generate more revenue accordingly. An important consequence of this diversification however, is that it introduces a non-trivial optimization problem to the provider with respect to the allocation of its available data center capacity to each pricing plan. In general, despite all benefits of reservation plan, computing resources provisioned by reservation plan generate lower revenue than that of on-demand plan. The spot market is also a choice for selling the spare capacity in the data center. Therefore, in this work, we address a novel problem of maximizing revenue through an optimization of capacity allocation to each pricing plan by means of admission control for reservation contracts, in a setting where aforementioned plans are jointly offered to customers. We devise both an optimal algorithm based on a stochastic dynamic programming formulation and two other algorithms that trade-off optimality and computational complexity. Our evaluation, which relies on an adaptation of a large scale real-world workload trace of Google, shows that our algorithms can significantly increase revenue compared to an allocation without capacity control given that sufficient resource contention is present in the system. In addition, we demonstrate that our algorithms effectively allow for online decision making and quantify the revenue loss caused by their assumptions to render the optimization problem tractable.

5.1 Introduction

IN the Infrastructure as a Service (IaaS) model of cloud computing, customers purchase units of computing time in the form of virtual machine (VM) instances in a flexible *pay-as-you-go* manner through the Internet [15]. Cloud providers maintain large-scale data centers to offer these computational resources on-demand and at a relatively low cost thanks to the economies of scale. Yet, to ensure business success, they need to obtain the highest possible revenue from selling available resource capacity. Methods such as adopting differentiated pricing plans, market segmentation [127] and demand forecasting [139], can be used so that the maximal amount of capacity is sold at the highest possible price.

As the computational resources involved constitute a non-storable or perishable commodity [134], cloud providers benefit from maximizing resource utilization in order to maximize revenue. Consequently, many IaaS providers offer various pricing plans (or markets) such as *reservation (subscription)* and *spot market*-based plans, in addition to an *on-demand pay-as-you-go* pricing plan. In the reservation plan, users pay an upfront reservation fee (premium) to reserve resources for a specific period of time (e.g., one year). In exchange, they receive a significant discount on the hourly resource usage price. The spot market allows users to submit the maximum price they are willing to pay to an auction-like mechanism as a bid. Users gain access to the acquired resources as long as their bid exceeds the provider's last computed market clearing price, which also determines the resource's uniform usage price until the next market clearing.

The segmentation of demand through different pricing plans is attractive to the provider for a number of reasons. For instance, risk-free income from reservations, on one hand, provides guaranteed cash flow through long-term commitments and can compensate for the demand uncertainty associated with the on-demand pay-as-you-go pricing plan [127]. The spot market, on the other hand, can attract price-sensitive users that are capable of tolerating service interruptions, allowing the provider to generate additional revenue from spare capacity in the data center without being exposed to the risks resulting from overbooking capacity.

The use of multiple pricing plans introduces a number of non-trivial trade-offs to IaaS providers with respect to revenue maximization. Although on-demand pay-as-you-go requests often generate the highest revenue per resource hour sold, they suffer from future demand uncertainty. The upfront fee of the reservation plan is beneficial from a cash flow perspective, but in the long-run, the total revenue generated is lower than the one obtained by selling equivalent usage hours under an on-demand pay-as-you-go plan. Moreover, the provider is liable to offer guaranteed availability for reserved requests to honor the associated Service Level Agreement (SLA), which might be costly when customers do not fully utilize their reserved capacity in the reservation's lifetime [4]. Allocating this underutilized reserved capacity to on-demand requests (i.e., overbooking resources), creates the risk of SLA violations. Spot instances on the other hand, can be terminated by the provider whenever their resources are required to honor commitments made with respect to other pricing plans. The provider therefore has the freedom to accommodate spot instances in the underutilized reserved capacity of the data center. Consequently, the benefits of this flexibility from a revenue management perspective must be considered when admitting new reservation contracts.

We address the problem of maximizing revenue when these three pricing models are jointly applied. Our main research question is the following: with limited resources available, and considering the dynamic and stochastic nature of customers' demand, how can expected revenue be maximized through an optimal allocation of capacity to each pricing plan?

We frame our algorithmic contributions within a *revenue management* framework that supports the three presented pricing plans and that incorporates an admission control system for requests of the reservation plan. To the best of our knowledge, we are the first to consider on-demand pay-as-you-go, reservation and spot market-based plans jointly offered in a revenue maximization problem of IaaS cloud providers.

In summary, our **main contributions** are:

- We formulate the optimal capacity control problem that results in the maximization of revenue as a finite horizon *Markov decision process (MDP)* [94]. We present a stochastic dynamic programming technique to compute the maximum number of reservation contracts the provider can accept from the arriving demand in order to maximize revenue. For a large capacity provider, the use of the stochastic dynamic programming technique is computationally prohibitive. We therefore present two algorithms to increase the scalability of our solution. The first increases the spatial and temporal granularity of the problem in order to solve it in a time suitable for practical online decision making. The second sacrifices accuracy to an acceptable extent through a number of simplifying assumptions on reserved capacity utilization and the lifetime of on-demand requests to increase scalability.

- We evaluate our proposed framework through large-scale simulations, driven by cluster-usage traces that are provided by Google. We propose a scheduling algorithm that generates VM requests based on the user resource usage in these traces. Under pricing conditions that are aligned with those of Amazon EC2[1], we demonstrate that our admission control algorithms substantially increase revenue for the provider.

This chapter is organized as follows: after reviewing the related work in Section 3.2, we introduce the system model in Section 5.3. Its subsections discuss 1) the cloud pricing models used in this chapter, 2) the optimal revenue management problem, and 3) a stochastic dynamic programming technique to tackle the problem. We propose the admission control algorithms namely *pseudo optimal* and *heuristic* in Section 5.4. Section 5.5 focuses on the revenue management framework and its architecture. The performance evaluation of the framework and a comparison between the admission control algorithms is presented in Section 5.6. Our conclusions are presented in Section 3.6.

5.2 Related Work

Revenue management (*also known as yield management*) is the process of maximizing revenue from a fixed, perishable resource capacity using market segmentation and demand management techniques. During the last few decades, revenue management has witnessed significant scientific and practical advances especially in the airline and hotel industries. The literature is vast on the topic and it is not our aim to cover the area exhaustively, but to rather present the relevant existing applications of this field to cloud computing. Interested readers can find a detailed overview of revenue management in [132].

One of the early attempts to incorporate revenue management into cloud computing was made by Püschel and Neumann [93]. They investigate the use of a policy-based

[1]`http://aws.amazon.com/ec2/`

admission control model to resource management components using techniques such as client classification and dynamic pricing. Similar work has been done by Meinl et al. [64] who applies derivative markets and yield management techniques for revenue maximization.

Macías et al. [60] propose several techniques such as dynamic pricing, over-provisioning, and selective SLA violation to maximize cloud provider revenue. Recently, Kashef et al. [50] proposed a system architecture for cloud service providers that combines demand-based pricing with resource provisioning. They compare two revenue management techniques for cloud computing. The first sets the timing for offering price discounts, whereas the second determines the number of VMs that should be offered at full price.

Anandasivam et al. [4] utilize a *bid-price control* technique that originates from the revenue management literature for capacity management which accepts or denies incoming requests for service in order to increase revenue. Bid-price control is an accepted and efficient method in Airline revenue management in which threshold values, which also called bid prices, are set for each leg of an itinerary and ticket is sold if its fare exceeds the sum of the bid prices along the path. Their model considers multiple resources such as CPU, memory, storage, and bandwidth, while our model comprises bundles of resources, i.e., VM instances.

The main difference between these works and ours is that none of them considers the joint adoption of the multiple different pricing plans presented in this chapter. As a result they are not applicable for most current cloud providers that are offering different pricing plans.

In our model, the provider is faced with stochastic and dynamic arrivals and departures of customers' requests and must decide on whether to admit an incoming reservation contract or to reject it. Similarly, Nair and Bapna [78] introduced a revenue management technique based on the admission control for the application domain of an Internet Service Provider (ISP). They formulate the problem as a continuous time Markov decision process over an infinite planning horizon to dedicate ISP capacity to customers at any instant of time. Despite these similarities, their application domain differs from cloud computing and their approach cannot be directly applied in the cloud context.

Mazzucco and Dumas [63] examine the problem of allocating servers to two classes of customers, *premium* and *basic*, in a way that maximizes provider revenue. The authors rely on a queuing model to tackle the optimization problem. Their work differs from ours since they have done the optimization for a case of a platform as a service provider; therefore their assumptions, pricing plans, and experimental setting differ.

There is a large body of research devoted to minimizing cost for cloud consumers using different pricing models, see for example [21,45,121,129]. However, limited investigation has been done on resource allocation and capacity planning techniques to maximize provider revenue. The problem of dynamically allocating resources to different spot markets for revenue maximization has been investigated by Zhang et al. [139]. Supply adjustment and dynamic pricing are used as a means to maximize revenue and meet customer demand. They model the problem as a constrained discrete-time and finite-horizon optimal control problem and adopt *Model Predictive Control* (MPC) techniques to design the dynamic algorithm solution. MPC is a widely used industrial technique for advanced multivariable control in the presence of nonlinearities and uncertainties. The study does not consider the coexistence of multiple markets, focusing solely on the spot market. Deciding on the optimal capacity segmentation for on-demand and spot market requests has been formulated as a Markov decision process by Wang et al. [127]. As a part of their work, they propose an optimal mechanism for a spot market based on the uniform price auction. In their model, they only consider on-demand pay-as-you-go and spot market requests and assume that reservation contracts are always fulfilled in highest priority.

Xu and Li [134] present an infinite horizon stochastic dynamic program to maximize the cloud provider's revenue with stochastic demand arrivals and departures.They aim at maximizing revenue specifically for the spot market and do not consider the case that the provider offers other pricing plans. Similarly, Truong-Huu and Tham [118] formulate the competition among cloud providers as a non-cooperative stochastic game which is modeled as Markov decision processes to maximize cloud providers' revenue. They provide dynamic resource pricing in which providers propose optimal price policies with

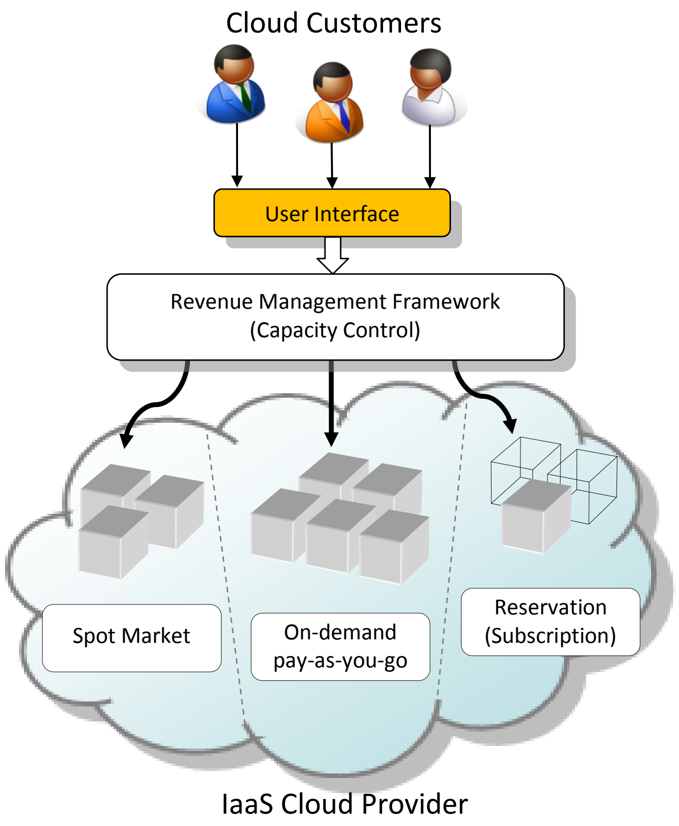


Figure 5.1: Schematic system model addressing the capacity control problem.

regard to the current policies of other competitors. They also introduce a novel approach for the cooperation among providers to enhance revenue and acquire the needed resources at any given time. Both studies rely on dynamic pricing as the main technique to maximize revenue, whereas our work focuses on capacity management and admission control without imposing any particular dynamic pricing policies.

5.3 System Model

In this section, we review common cloud pricing plans, and formulate the optimal capacity control technique with a revenue maximization objective. Fig. 5.1 shows the schematic system model.

Table 5.1: Pricing of the standard small on-demand, reserved and spot instances (`m1.small`, Linux, us-east) in Amazon EC2[2]

Pricing Plan	Upfront	Hourly rate
On-demand	\$0	\$0.060
1-year Reserved (light Utilization)	\$61	\$0.034
1-year Reserved (Medium Utilization)	\$139	\$0.021
1-year Reserved (Heavy Utilization)	\$169	\$0.014
Spot	\$0	Spot Market Price

5.3.1 Cloud Pricing Plans

On-demand pay-as-you-go plan

This plan charges customers for compute capacity based on actual usage, without requiring any contractual long-term commitments. The service is charged for at a fixed rate per billing cycle (e.g., hourly) from the time the instance is launched until it is terminated. Given an hourly price of p the customer is charged ph for an instance held for h hours. After instantiation of an instance, customers can retain it for as long as they wish. A request for on-demand instances can be denied if the provider has insufficient resources available. Note that the rate for on-demand pay-as-you-go instances is fixed at most IaaS providers for a long period of time (i.e., months to years), and can therefore be viewed as a constant value. Moreover, the one-hour billing cycle, selected based on the Amazon EC2 billing period, can be replaced with any other billing period, e.g., per-minute or per-day billing cycle without any specific change in our model.

Reservation plan

This plan allows customers to reserve an instance for a certain *reservation period* (e.g., months or years) and assures that the reserved capacity is available whenever it is required in that period. During the reservation period, the reservation is said to be *live*.

The customer must pay an upfront *reservation fee* (premium) of φ, after which the usage is either free (e.g., as in GoGrid[3]) or heavily discounted (e.g., Amazon EC2). The

[2]Amazon EC2 pricing as of March. 30, 2014, `http://aws.amazon.com/ec2/pricing/`.
[3]GoGrid, `http://www.gogrid.com/`.

premium is a one-time fee that must be paid irrespective of how much the instance is used during the reservation period. The total amount of instance hours consumed by a single customer account are aggregated per billing cycle and then automatically matched to any reserved capacity contracts the customer has in its portfolio.

Let $\alpha \in [0, 1]$ be the discount factor on the on-demand plan's rate that is obtained when reserving a given instance type. Then, total h hours of running the reserved instance in the whole reservation period costs $\varphi + \alpha p h$. For example, in Amazon EC2, a premium of \$61 reserves an `m1.small` instance (Linux, US East, Light Utilization) for 1 year, resulting in a \$0.034 per hour usage price within the reservation period compared to the on-demand hourly rate of \$0.060 ($\alpha = 0.57$), see also Table 5.1. Partial utilization of the reserved capacity can still lead to cost benefits for customers. For example, for the `m1.small` reserved instance, a cost reduction is obtained if the instance runs for more than 2347 hours (or roughly 98 days ¡¡ 1 year), that is, $61 + 2347 \times 0.034 \simeq 2347 \times 0.060$. Therefore, the *break-even point* is 98 days and reservation is only beneficial if it is used more than roughly 98 days.

Some cloud providers offer multiple reservation plans with different reservation periods and expected utilization levels. For example, Amazon EC2 offers 1 or 3-year terms contract for light, medium, and heavy levels of utilization. For the sake of simplicity, we limit the discussion to one type of reservation only within a given reservation period (τ) in number of time units (e.g., hours). Our model can be extended to include more than one type of reservations.

Spot market

In this plan, customers submits their bids for acquiring instances and subsequently the provider reports a market-wide clearing price at which instances are charged. The instance can be terminated by the provider as soon as the spot market's clearing price rises above the customer's bid. The customer therefore does not have solitary control over the instance's lifetime.

The provider can use a variety of market mechanisms for the spot pricing, e.g., variants of auction mechanisms that determine the allocation and pricing rules. Likewise, the frequency of the mechanism's clearing can be varied (e.g., upon each bid arrival, instance termination, every hour). At present, providers offer limited transparency with respect to the actual mechanisms used in their spot markets.

Consequently, we do not consider any specific spot pricing mechanism in this work, and situate the fine-grained computation of spot price dynamics outside the scope of this work. Instead, we model spot instances' price by a constant factor $\beta \in [0, 1]$ that determines the average discount rate for these instances' hourly usage relative to the on-demand rate. We therefore assume that on average, the spot market price lies below the on-demand rate, which is reasonable given the lower quality of service (QoS) provided. According to Amazon EC2, recent spot prices on average are typically 86% lower compared to on-demand pay-as-you-go instances, i.e., $\beta = 0.14$. We assume that provider always retains the capability of terminating spot instances in favor of more profitable requests as a strategic tool to increase revenue.

In general, on-demand pay-as-you-go instances can generate more revenue for the provider, while reserved instances can provide a risk-free upfront income and foster long term commitments by customers. A disadvantage of the reservation plans is that the provider is liable to provide guaranteed availability for the reserved requests while customers do not necessarily fully utilize their reserved capacity in the reservation period. An opportunity therefore exists to make this underutilized capacity available to instance requests originating from other pricing plans. As spot instances are allowed to be terminated by the provider, we model the possibility that the provider accommodates spot instances in the underutilized reserved capacity of the data center. In principle, it is also possible to make underutilized capacity available to on-demand instance requests. This however, creates the risk of SLA violations occurring as the provider has no direct control over the lifetime of on-demand instances. In this chapter, we do not allow accommodating on-demand requests in underutilized reserved capacity, and therefore rule out this further opportunity for revenue maximization.

5.3.2 The Optimal Capacity Control Problem

To maximize revenue, the cloud provider aims to optimally allocate its available capacity to requests from different pricing plans. In this section, we formally describe the problem of optimizing admission decisions on reservation contracts such that the overall revenue is maximized.

Suppose that the provider's capacity is C for a specific instance type, i.e., at any given time, up to C instances of that type can be hosted simultaneously. We consider the given instance type as the only one in the system. Consequently it represents our unit of capacity. However, this is not a limiting assumption as we can model other instances as multiples of the unit capacity with a limited error.

We discretize the time horizon into identically sized slots. The slot size is aligned with the provider's billing cycle (e.g., an hour). We assume that, given the large degree of workload multiplexing, the provider is able to predict upcoming demand for its different pricing plans for Γ time slots[4]. Suppose that at the current time $t = 0$, the provider predicts the number of requests in the reservation, on-demand and spot markets for a window of size Γ as $(d^r_0, ..., d^r_{\Gamma-1})$, $(d^o_0, ..., d^o_{\Gamma-1})$ and $(d^s_0, .., d^s_{\Gamma-1})$ respectively. The provider makes a decision to admit r_t reservation contracts at time t with $0 \leq r_t \leq d^r_t$ to maximize the revenue generated in the window. Our formulation is therefore greedy with respect to the size of the prediction window.

Let l^r_t denote the total number of previously admitted reservation contracts that remain live at time t (i.e., reserved capacity at time t is $l^r_t + r_t$). Similarly, the total number of previously running on-demand and spot instances that remain active at that time are denoted by l^o_t and l^s_t respectively. Therefore at time t the provider can accommodate the maximum of o_t additional on-demand instances without overbooking the infrastructure, as per following equation:

$$o_t = \min(C - l^r_t - r_t - l^o_t, d^o_t) \tag{5.1}$$

Let $u_t \in [0, 1]$ denote the utilization of the reserved capacity at time t, e.g., if the

[4]Note that our aim, in this chapter, is not to present specific workload prediction techniques and this has previously been addressed in the literature [86, 139].

total number of live reservations at time t is 1000 and 600 reserved instances are running at that time, $u_t = 0.6$. After accommodating the reservation contracts and on-demand pay-as-you-go requests, the remaining capacity can be used for spot instances, that is, $\min(C - u_t \times (l_t^r + r_t) - l_t^o - o_t, d_t^s)$ spot instances can be accommodated, where $u_t \times (l_t^r + r_t)$ represents total number of reserved instances running at time t.

Problem definition: The provider's problem is to find $r_0, r_1, ..., r_{\Gamma-1}$, or, in other words, to decide how many reservation contracts must be admitted for each time slot such that the revenue obtained within the prediction window is maximized. The total revenue that can be obtained within the window is:

$$\sum_{t=0}^{\Gamma-1} r_t \varphi + \alpha p u_t (l_t^r + r_t) + p(l_t^o + o_t) + \beta p (l_t^s + s_t)\,, \tag{5.2}$$

where the first term is the revenue from the upfront reservation fees and the second, third and fourth terms are the revenues per time slot from running reserved, on-demand, and spot instances respectively. The maximization problem can therefore be defined as:

$$\begin{aligned} \max_{r_t} \sum_{t=0}^{\Gamma-1} r_t \varphi + \alpha p u_t (l_t^r + r_t) + p(l_t^o + o_t) + \beta p (l_t^s + s_t) \\ \text{s.t} \qquad l_t^r + r_t + l_t^o + o_t \le C\,, \\ u_t(l_t^r + r_t) + l_t^o + o_t + l_t^s + s_t \le C\,, \\ \forall t = 0, ..., \Gamma - 1\,. \end{aligned} \tag{5.3}$$

Here, the first constraint ensures that the number of live reservations and running on-demand instances remains within the provider's capacity, thereby ensuring that no SLA violations on the reservation contracts can occur. The second constraint limits the total amount of running instances over all pricing plans to the same capacity.

The optimization problem (5.3) is non-trivial and by no means easy to solve. The root cause of the problem's complexity lies in the fact that the number of running instances in each slot for each pricing plan depends on the history of admitted requests in previous slots. Moreover, the duration that instances remain active in the system is not known a priori as the provider is often unaware of the application type running in the instance.

In the next section, first we make a few assumptions that make the problem tractable and reduces the problem's complexity. Afterwards, we propose a stochastic dynamic programming solution to tackle problem (5.3). For reference, Table 5.2 summarizes the symbols used throughout the chapter and their definitions.

5.3.3 Optimal Capacity Control

We devise a stochastic dynamic programming formulation to tackle problem (5.3) in this section. First, we introduce a number of assumptions made for solving the optimization problem. After that, using a set of recursive *Bellman* equations [94], we show that the problem can be broken down into simpler sub-problems, each of which can be solved optimally. Finally, we present our stochastic dynamic programming approach, while optimal is computationally prohibitive to use for cloud providers of large scale.

Assumptions

In general, the lifetime of on-demand pay-as-you-go instances is not known to the provider in advance. By lifetime, we mean the amount of time the customer runs the instance from the time it is started until it is stopped or terminated. We denote by h_j the lifetime of instance j. To make the analysis tractable, similar to [127] we assume that h_j's are exponentially i.i.d. (Independent and Identically Distributed) random variables. In our discrete settings, this means that h_j follows a *geometric distribution* [94] with a probability mass function (pmf) of $P(h_j = k) = (1-q)^{k-1}q$ for $k = 1, 2, 3, ...$, where q is the probability that the customer terminates the currently running instance in the next time slot. Since the expected value of h_j is $1/q$, the expected payment over the lifetime of an on-demand pay-as-you-go instance is $E[h_j p] = p/q$.

In practice, the spot market's underlying market mechanism must be run at each time slot, involving bids from newly arrived requests and currently running spot instances. In fact, the provider does not distinguish newly submitted requests and those requests that are admitted previously in each round of auction [127]. Moreover, spot instances can be terminated by the provider at any time during their execution by adjustment of the mar-

Table 5.2: Symbols and Definitions.

Symbol	**Definition**
Γ	Prediction window size
p	On-demand pay-as-you-go instance price per hour
φ	Upfront reservation fee (premium)
α	Discount rate due to reservation, the reserved instance price is αp per hour
β	Ratio of average price of spot to on-demand instances
r_t	Number of reservation contracts admitted at time t
o_t	Number of on-demand pay-as-you-go instances accepted at time t
s_t	Number of spot pay-as-you-go instances accepted at time t
d_t^r	Predicted number of reservation contracts at time t
d_t^o	Predicted number for on-demand pay-as-you-go requests at time t
d_t^s	Predicted number of spot instances at time t
l_t^r	Total number of live reservation contracts at time t
l_t^o	Total number of active on-demand instances at time t
l_t^s	Total number of active spot instances at that time t
τ	Reservation period in number of time slots
u_t	Utilized reserved capacity by reserved instances at time t
i_t	Reserved capacity utilization class interval to which the reserved capacity utilization at time t belongs
$\|Z\|$	Number of reserved capacity utilization class intervals
z_i	Representative value of the class interval i
h_j	lifetime of instance j in number of hours
ς_t	Data center state at stage t, $\varsigma_t = (l_t^r, l_t^o, i_t)$
q	Termination probability of the running on-demand pay-as-you-go instance in the next time slot
λ_t	Discount factor for upfront reservation fee at time t
$\bar{u}$	Mean reserved capacity utilization
e_t^r	Number of expired reservations by the end of time t
$V(\varsigma_t)$	Expected revenue obtained from $t = 0$ to $\tau - 1$
$P(\varsigma_{t+1} \mid \varsigma_t, r_t)$	Transition probability from ς_t to ς_{t+1} given the chosen action r_t
$\gamma(\varsigma_t, r_t)$	The revenue for each state-action pair
B	Number of instances per block of capacity
T	Number of billing cycles (hours) per each time slot

ket clearing price. This allows the provider to shape the load according to the available capacity and user bids. Therefore, to avoid the resulting complexity with respect to the lifetime of spot instances, we assume that the load for the spot market in each time slot is independent of the previous slots and is solely defined by demand on that time slot (d_t^s), i.e., $l_t^s = 0$. The load prediction component in our framework therefore computes d_t^s based on the aggregated load of the spot market in past time slots, that is, d_t^s implicitly includes l_t^s.

We treat u_t, the reserved capacity utilization at time t, as a categorical random variable. We categorize the reserved capacity utilization range into a set of $|Z|$ classes. Each class is associated to a utilization interval, denoted by i, of which the midpoint is used as the representative value of the corresponding class. The representative value of the i'th class interval is denoted by $z_i \in Z$ with $0 \leq i < |Z|$. For instance, if we take $|Z| = 5$, the utilization range of $[0, 1]$ is divided to five class intervals of $[0, 0.2]$, $[0.2, 0.4]$, ..., $[0.8, 1]$. $Z = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ is used as a set of discrete values for categorizing the reserved capacity utilization. If u_t lies within $[0.2, 0.4]$, then it belongs to the class interval 1 and the class interval representative value of 0.3 is used as utilization value at time t. Note that the class interval to which the reserved capacity utilization at time t belongs is denoted by i_t, and in our calculation, we use representative value of that class as the reserved capacity utilization at that time (i.e., z_{i_t}).

Treating u_t as a discrete random variable is necessary for the dynamic programming solution we propose in this section and the number of class intervals can be chosen depending on the desired granularity of the analysis. The provider is assumed to have sufficient data center load history available in order to derive the pmf of u_t in advance, i.e., $P(u_t = z_{i_t})$ is known for all z_{i_t}.

Suppose ς_t denote the state at time t that holds all information about the load in the data center at time t. In order to find an optimal solution for problem (5.3) using dynamic programming, each state ς_t at time t is required to depend solely on the state at time $t-1$ (ς_{t-1}) and be independent of all earlier states $\varsigma_{t-2}, \varsigma_{t-3}, ..., \varsigma_0$. As we know, every state ς_t must include the total number of live reservations at time t. Clearly, with a reservation period of size τ, the total number of live reservations at time t depends on $r_{t-\tau+1}, ..., r_{t-1}$, as reservations admitted earlier than $t - \tau + 1$ will no be longer available at t. To make ς_t only dependent on ς_{t-1}, one could resort to the inclusion of $\tau - 1$ values in the state, each one representing the number of instances reserved at $t - i, i = \tau - 1, ..., 1$. This leads to a high-dimensional state space. Note that τ is often large (e.g., the number of hours in one year) and the number of instances that are reserved at each time slot t can be as large as C. Iteration over the possible states in the problem space therefore results in exponential time complexity, leading to the *curse of dimensionality* [90].

In practical online cases, the provider is interested in finding the admission threshold at the current time instantly. Moreover, the impact of admitting a reservation at t is only affected by future events in the reservation period $[t, t + \tau - 1]$. We therefore limit the prediction window $\Gamma = \tau$. This significantly reduces the dimensional space of each state as every admitted reservation in the window remains live until the end of the prediction window. Therefore, state ς_t only encompasses the total number of live reservations at time t which only depends on l_t^r and r_t the number of accepted reservation contracts in that state.

Dynamic Programming Formulation

We start our formulation by defining *stages* (decision epochs) and *states*. The decision problem consists of τ stages indexed 0 to $\tau - 1$, each representing a time slot. The provider must decide on the number of admitted reservation contracts (r_t) at each time slot t to maximize revenue.

A state at stage t is denoted by $\varsigma_t = (l_t^r, l_t^o, i_t)$. ς_t holds all information about the load in the data center at time t, i.e., the number of reservations that remain live from previous time slots in time slot t (l_t^r) and the total number of running on-demand (l_t^o) instances. The number of running reserved instances can be computed based on z_{i_t} as well. The number of spot instances is bounded by the available capacity or the spot market demand:

$$s_t = \min(C - (l_t^r + r_t)z_{i_t} - (l_t^o + o_t), d_t^s) \,. \tag{5.4}$$

The provider must decide to perform one of the possible *actions* to admit r_t reservation contracts at stage t, with $0 \leq r_t \leq d_t^r$.

We define λ_t as a discount factor that linearly scales the reservation fee with respect to the remaining time until the end of the prediction window. This measure is required as the prediction window is taken to be as large as the reservation period, which in itself is required for making sound optimization decisions. A reservation admitted at time t expires at time $t + \tau - 1$, which for all $0 \leq t < \tau$ total $\tau - t$ time slots of which are within the window period. Therefore, we apply a discount on the premium fee (φ) proportional to the effective reservation period in the window. In each stage t, we thus define $\lambda_t = (\tau - t)/\tau$ with $0 \leq t < \tau$.

In our model, the amount of the revenue obtained by the provider in each stage depends on the current state (ς_t) and the provider's choice for r_t. The revenue of each state-action pair is defined as:

$$\gamma(\varsigma_t, r_t) = \lambda_t r_t \varphi + \alpha p(l_t^r + r_t) z_{i_t} + p(l_t^o + o_t) + \beta p s_t \,, \tag{5.5}$$

where the first, second, third and fourth term are the total revenue of reservations, reserved, on-demand, and spot instances respectively.

Suppose there are n running on-demand pay-as-you-go instances in the data center at time $t-1$ (i.e., $n = l_{t-1}^o$) and right before t, X of them are terminated by customers. There are $l_t^o = n - X$ active instances remaining at the beginning of t. According to the assumption of the geometric lifetime of on-demand instances, one can see that X follows a *binomial distribution* [94] with $P(X = k) = Bin(k; n, q)$, where $Bin(k; n, q) = \binom{n}{k} q^k (1 - q)^{n-k}$. Here, q is the probability that the running on-demand instance is terminated in the next time slot.

As stated before, each admitted reservation within the window remains active until the last stage ($t = \tau - 1$). However, at the beginning of each time slot t, some reservations expire as they are admitted before time $t = 0$. We define e_t^r as the number of reservations that are expired by the end of t. Therefore, for a window of size τ, $(e_0^r, ..., e_{\tau-2}^r)$ encodes all information regarding expired reservations in each stage. $(e_0^r, ..., e_{\tau-2}^r)$ can easily be obtained based on the provider's history of admitted reservation contracts.

From the above discussion, it follows that ς_{t+1} can be computed based on ς_t only.

In fact, total number of live reservations at time t solely depends on l_t^r, e_t^r and r_t by the relation $l_{t+1}^r = l_t^r + r_t - e_t^r$.

From the *memorylessness*[5] property of the geometric distribution, l_{t+1}^o can also be easily computed only using the previous state. Finally, according to the definition, $z_{i_{t+1}}$ is independent of z_{i_t}. Therefore, we make an important observation, that state ς_{t+1} only depends on state ς_t at the previous time and is *independent* of earlier states $\varsigma_0, ..., \varsigma_{t-1}$.

Let us define $V(\varsigma_t)$ as the expected revenue obtained from $t = 0$ to $\tau - 1$. The problem of revenue maximization through optimal admittance of reservation contracts can be characterized by the following *Bellman equations* [94]:

$$V(\varsigma_t) = \max_{r_t}[\gamma(\varsigma_t, r_t) + \sum_{\varsigma_{t+1}} P(\varsigma_{t+1}|\varsigma_t, r_t)V(\varsigma_{t+1})]\,. \tag{5.6}$$

In (5.6), the maximum revenue the provider can obtain at state ς_t by optimally choosing r_t is given by the expected maximum revenue over all possible states ς_{t+1}. The boundary conditions of (5.6) are given by $V(\varsigma_\tau) = 0$ for all ς_τ. $P(\varsigma_{t+1}|\varsigma_t, r_t)$ represents the transition probability to ς_{t+1} given state ς_t and action r_t. Given $k = (l_t^o + o_t) - l_{t+1}^o$, the desired transition probability is computed as follows:

$$P(\varsigma_{t+1}|\varsigma_t, r_t) = P(u_{t+1} = z_{i_{t+1}}) \times Bin(k; l_t^o + o_t, q)\,, \tag{5.7}$$

where $P(u_{t+1} = z_{i_{t+1}})$ shows the probability that the reserved capacity utilization at stage $t + 1$ falls in the class interval i_{t+1} and $Bin(k; l_t^o + o_t, q)$ shows the probability that k on-demand instances are terminated in a transition from ς_t to ς_{t+1}. Since these two events are independent, the probability of both occurring is the product of the probabilities. Note that the probability of change in reserved capacity from l_t^r to l_{t+1}^r is equal to 1 given the exact value of r_t. That is, it is known how many of reservation contracts will expire at the end of time slot t based on the admittance history. The above analysis converts problem (5.3) into a dynamic programming problem (5.6).

[5]In probability theory, memorylessness is a property of those distributions (e.g., the exponential distributions and the geometric distributions), wherein any derived probability from a set of random samples is distinct and has no information of earlier samples.

Complexity of Optimal Capacity Control

Equation (5.6) represents a Markov decision process that can be solved by numerical dynamic programming through backward induction. It commences the search for a solution by simulating the load for each pricing plan (market) based on the predicted demand in the last stage $\tau - 1$ and calculating the optimal number of reservations that must be admitted in that stage. Using results for the last stage, it then proceeds to determine the optimal solution for the previous stage (*backward induction*). This process continues until the optimal solution at stage $t = 0$ is obtained.

The number of possible actions at each stage is at most $C + 1$ taking into consideration $d_t^r \leq C$. The number of possible states at stage t, i.e., $|\varsigma_t|$ is at most $(C + 1)^2 \times |Z|$ since $0 \leq l_t^r, l_t^o \leq C$. In each stage t, the maximization must be done over every possible action for all states which by itself requires a computation of expected revenue over all possible states at stage $t + 1$. Therefore, the complexity of a single-stage calculation is $O(C \times (C^2 \times |Z|)^2)$. As there are τ stages, the overall computational complexity is $O(\tau \times C^5 \times |Z|^2)$.

For IaaS cloud providers with large capacity (e.g, $C = 10^5$) and long reservation period (e.g., $\tau = 1$ year) finding exact solution to the (5.6) is computationally prohibitive as decisions need to be made in real time. However, solving problem (5.6) at the granularity of a single VM and a billing cycle of an hour is not essential for large cloud providers with a massive amount of cash flow. Thus, we define our *pseudo optimal* algorithm based on larger blocks of capacity and time which approximates the optimal solution and can solve the problem in a reasonable time. We also propose a *heuristic* algorithm which significantly reduces the time complexity at the price of sacrificing a fraction of the revenue.

5.4 Proposed Algorithms

5.4.1 Pseudo Optimal Algorithm with an Efficient Computational Time

In order to overcome the complexity of the proposed stochastic dynamic programming technique, we present a *Pseudo Optimal* heuristic that allows for a reduction in dimensions of the problem. Suppose B be the number of VM instances per block of capacity (e.g.,

$B = 100$ *VMs*) and T be the number of billing cycles per time slot (e.g., $T = 168$ *hours*). We apply the same approach presented in subsection 5.3.3, while increasing the granularity of the problem formulation with respect to capacity and time. We therefore map the values of the original problem variables onto representative values given the chosen block sizes and use these in Algorithm 1 to find the solution. For example, for $B = 100$, all capacity values are rounded to the nearest multiple of 100. On line (17) of Algorithm (1), the revenue of each state-action pair is therefore scaled in terms of T and B. Note that all previously used notations related to the capacity or time must be interpreted in multiples of B and T, e.g., if $T = 24$ *hours* and the reservation period is 365×24 *hours* (365 days) then $\tau = 365$. Likewise, if $B = 100$ and the total number of reservations that remain live at time t equals 500 then $l_t^r = 5$.

Reducing the granularity of the optimization problem not only reduces the problem size but also removes the necessity for accurately predicting future demand at the fine-grained level of VMs and billing cycles.

5.4.2 Heuristic Algorithm with a Low Computational Complexity

The pseudo optimal algorithm proposed in the previous section can be run in an acceptable time frame if T and B are taken to be sufficiently large. But it still suffers from the prohibitively high polynomial order for small values of T and B (e.g., an hour and one VM). Therefore, we propose our *heuristic* algorithm with a lower computational time complexity that can find an approximated solution quickly for any values of T and B.

The idea behind the heuristic algorithm is that whenever the provider admits a reservation it might require to reject upcoming future on-demand requests in order to fully guarantee the availability of the reserved instances. Clearly, the admission of a reservation contract is well justified *if and only if* the revenue loss due to rejections of on-demand instances does not exceed the total revenue the reservation generates. Two main factors can affect that revenue: 1) the utilization of the reserved capacity, and 2) the demand in the spot market. The more the reservation is utilized, the higher revenue it generates in total. As stated in earlier sections of this chapter, the provider is able to accommodate spot instances in the reserved capacity without any concern for the availability of the re-

Algorithm 1 Pseudo Optimal Algorithm

```
Input: $t, l_t^r, l_t^o, i_t$
Output: $maxrev$
1: dp $\leftarrow \{-1\}$ $\triangleright$ matrix dp is used for memoization and all cells are initialized with -1.
2: function V($t$, $l_t^r$, $l_t^o$, $i_t$)
3:     if dp$[t][l_t^r][l_t^o][i_t] \neq -1$ then
4:         return dp$[t][l_t^r][l_t^o][i_t]$
5:     end if
6:     if $t = \tau$ then
7:         dp$[t][l_t^r][l_t^o][i_t] = 0$
8:         return 0
9:     end if
10:    $maxrev \leftarrow 0$
11:    for $r_t \leftarrow 0$ to $\min(C - l_t^r - l_t^o, d_t^r)$ do
12:        $rev \leftarrow 0$
13:        $l_{t+1}^r \leftarrow l_t^r + r_t - e_t^r$
14:        $o_t \leftarrow \min(C - l_t^r - l_t^o - r_t, d_t^o)$
15:        $s_t \leftarrow \min(C - (l_t^r + r_t)z_{i_t} - l_t^o - o_t, d_t^s)$
16:        $\lambda \leftarrow (\tau - t)/\tau$
17:        $\gamma(\varsigma_t, r_t) \leftarrow B\lambda r_t \varphi + BT(\alpha p(l_t^r + r_t)z_{i_t} +$
                   $p(l_t^o + o_t) + \beta p s_t)$
18:        for $l_{t+1}^o \leftarrow 0$ to $l_t^o + o_t$ do
19:            for $i_{t+1} \leftarrow 0$ to $|Z|$ do
20:                $P(\varsigma_{t+1}|\varsigma_t, r_t) \leftarrow P(u_t = z_{i_{t+1}}) \times Bin(l_t^o + o_t - l_{t+1}^o; l_t^o + o_t, q)$
21:                $rev \leftarrow rev + \gamma(\varsigma_t, r_t) + P(\varsigma_{t+1}|\varsigma_t, r_t) \times \mathrm{V}(t+1, l_{t+1}^r, l_{t+1}^o, i_{t+1})$
22:            end for
23:        end for
24:        if $rev \geq maxrev$ then
25:            $maxrev \leftarrow rev$
26:        end if
27:    end for
28:    dp$[t][l_t^r][l_t^o][i_t] \leftarrow maxrev$
29:    return $maxrev$
30: end function
```

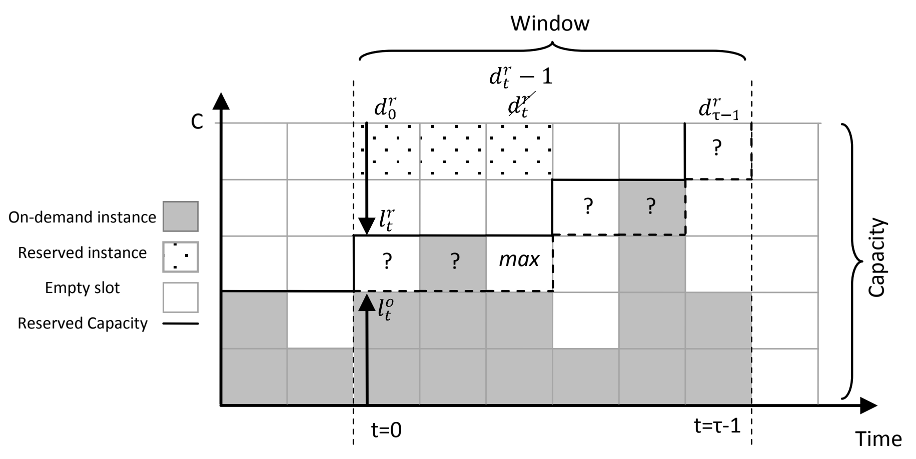


Figure 5.2: Illustration of Algorithm 2. Each small block shows the capacity unit per time unit (e.g., instance-hour). Schematically, reserved instances occupy the available capacity top-down and on-demand instances use the capacity bottom-up. For sake of simplicity, spot instances are not shown in the figure.

served instances, since spot instances can be terminated as the need arises. Therefore, if admission of a reservation provides capacity for accommodation of a spot request that might be rejected previously due to the lack of capacity, this additional revenue must be taken into account by the revenue management system.

The heuristic algorithm has two main simplifications compared to the *pseudo optimal* algorithm. First, instead of using the instance lifetime distribution to estimate load induced by on-demand requests, the future load is generated assuming all arriving requests pertain to instances with the same lifetime of mean value. Second, it relies on the average utilization of the reserved capacity $\bar{u}$ to control admission of reservation contracts. In fact, $\bar{u}$ is the expected value of the categorical pmf related to reserved capacity utilization.

Algorithm 2 presents the details of the proposed heuristic and Fig. 5.2 illustrates the operation of the algorithm. As we estimate the load in the on-demand pay-as-you-go market beforehand, with a slight abuse of notation, suppose l_t^r and l_t^o be the number live reservations (reserved capacity) and the number of running on-demand instances at time slot t respectively. l_t^o is computed according to the previously instantiated VMs (before time $t = 0$) and arriving demand (d_t^r) when each request has the mean lifetime. The shaded area in the bottom of Fig. 5.2 exemplifies such a load. Using e_t^r and history of

Algorithm 2 Heuristic Algorithm

```
Input: $l^r, l^o$    ▷ $l^r_t$ is the initial reserved capacity at time $t$ for those requests admitted before time $t = 0$. $l^o_t$ indicates the number of on-demand instances at time $t$ taking into account previously instantiated VMs, arriving demand, and assuming that every on-demand instance has the same lifetime of mean value.
Output: r
 1: function HEURISTIC($l^r$, $l^o$)
 2:     $r \leftarrow \{0\}$    ▷ Create the array $r$ with size of $\tau$ and initialize all elements with zero.
 3:     loop
 4:         $max \leftarrow -\infty, index \leftarrow -1, sum \leftarrow 0, sw \leftarrow 0$
 5:         for $t \leftarrow \tau - 1$ to 0 do
 6:             $l^r_t \leftarrow l^r_t + 1$
 7:             if $l^r_t > C$ then
 8:                 $l^r_t \leftarrow l^r_t - 1, sw \leftarrow 1,$ break
 9:             end if
10:             $ls \leftarrow 0$
11:             if $C < l^r_t + l^o_t$ then
12:                 $sum \leftarrow sum + T \times B(p\alpha\bar{u} - p)$
13:                 $ls \leftarrow d^s_t$
14:             else
15:                 $sum \leftarrow sum + T \times B(p\alpha\bar{u})$
16:                 $ls \leftarrow d^s_t - (C - l^r_t - l^o_t)$
17:             end if
18:             if $(l^r_t - 1) \times (1 - \bar{u}) < ls$ then
19:                 $sum \leftarrow sum + T \times B(1 - \bar{u})\beta p$
20:             end if
21:             $\lambda \leftarrow (\tau - t)/\tau$
22:             if $sum + B\varphi\lambda \geq max$ and $d^r_t > 0$ then
23:                 $max \leftarrow sum + B\varphi\lambda$
24:                 $index \leftarrow t$
25:             end if
26:         end for
27:         if $index = -1$ or $max < 0$ then
28:             break
29:         end if
30:         if $sw = 0$ then
31:             for $t \leftarrow 0$ to $index - 1$ do
32:                 $l^r_t \leftarrow l^r_t - 1$
33:             end for
34:         end if
35:         $r_{index} \leftarrow r_{index} + 1, d^r_t \leftarrow d^r_t - 1$
36:     end loop
37:     return $r$
38: end function
```

reservation contracts admitted earlier than $t = 0$, initial value of l_t^r is computed within the prediction window.

The algorithm attempts to admit as many reservation contracts as possible by filling the slots from the end of the window to the beginning (denoted by the question marks in Fig. 5.2). In each iteration, it adds one unit to l_t^r, computes the revenue this additional reservation generates and adds this to the sum of the total revenue (Lines 12 and 15). This computation takes into account the potential revenue that spot instances can generate as well. The spot market demand that must be accommodated in the underutilized reserved capacity is computed (ls) and its revenue which is proportional to the underutilized reserved capacity is added (Line 19), if the revenue of ls is not compensated by previously admitted requests.

If the admission of a reservation overlaps with on-demand pay-as-you-go load for the specific capacity block and time slot (see Fig. 5.2), the corresponding price of on-demand instances ($B \times T \times p$) must be deducted from the total revenue until this point (Line 12). Lines 22-25 keep track of the maximum revenue found thus far. The upfront reservation fee that is proportional to the effective part of the reservation period in the window is also taken into account in finding the maximum revenue value (max). After the maximum value and its corresponding time slot have been found, if there is available reservation demand on that time slot ($d_t^r > 0$) then the reservation contracts is admitted ($r_t = r_t + 1$) and d_t^r is reduced by one unit. The process finishes when the maximum value is negative (Line 27) or the reservation load exceeds the available capacity (Line 7).

The computational complexity of Algorithm 2 is $O(\tau \times C)$, as in the worst case all available slots in the window must be investigated. The heuristic is thus considerably more efficient than the pseudo optimal algorithm in terms of computational complexity which makes it suitable for online admission control.

5.5 Revenue Management Framework

In this section, we briefly discuss how Algorithms 1 and 2 can be practically implemented for online use in a real-world system. The algorithms are deployed in the admission con-

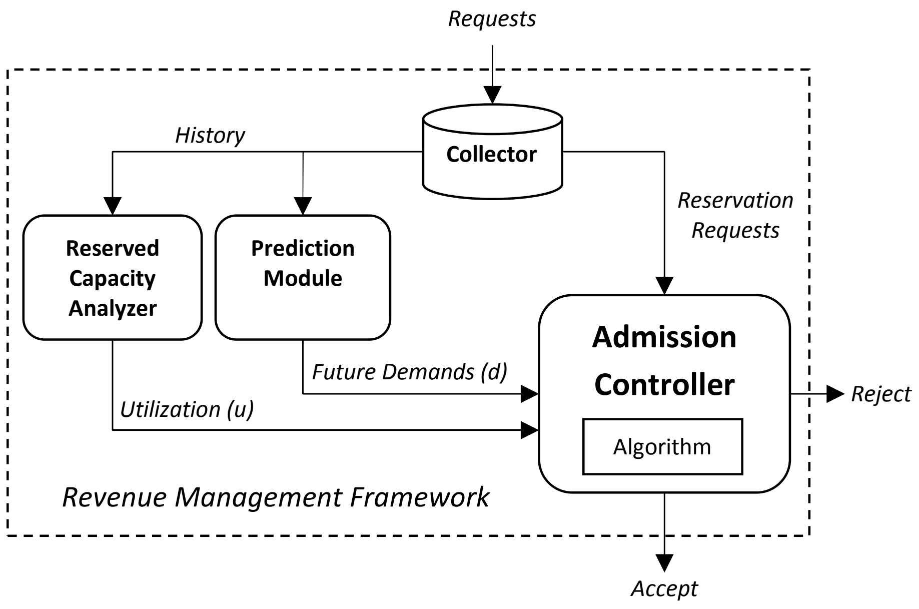


Figure 5.3: Key modules of the revenue management framework.

trol module of revenue management framework of which the key modules are illustrated in Fig. 5.3 and discussed next.

The *collector* collects and stores demand information for different markets (pricing plans). It also tracks the number of rejected requests for different markets. The collected information is used by the *prediction module* and the *reserved capacity analyzer* to be fed into the *admission controller*.

The main role of the reserved capacity analyzer is obtaining the categorical probability distribution of the reserved capacity utilization for the pseudo optimal algorithm or the expected value ($\bar{u}$) for the heuristic algorithm. In our implementation, the probability distribution is dynamically derived from the history of the data center load. During each time slot, the reserved capacity analyzer measures the period of time that the utilized reserved capacity falls into the different utilization class intervals introduced in Section 5.3.3. It then computes the probability of each utilization class interval occurring based on the statistics collected in each time slot. Eventually, the categorical pmf is generated by averaging the computed probabilities of the last τ time slots. We also use the expected value of the distribution to set $\bar{u}$ in case of the heuristic algorithm.

The prediction module forecasts future demands for a window of size τ for each market. Forecasting future demand is a well-studied area in the literature [86, 139] and it is beyond the scope of this chapter to present the best forecasting method here. Hence, we adopt a basic method for forecasting future demands, which can be replaced with a customized prediction method in practical implementations. In our model, the prediction module forecasts demands based on the history of the load in the data center in the sense that the predictor assumes the observed demands for past τ time slots would be repeated for future τ time slots.

For the reservation market, the number of reservation contracts received by the provider per time slot is rounded to the nearest multiple of B. A similar transformation is used for the demand in the on-demand market. For spot instances, the prediction module computes the average load per time slot and rounds it to the nearest capacity block representative value (B). That is, the area below the spot market's load curve is computed and divided by the slot time duration. The prediction module also incorporates the rejected demands into the predicted future demand using the number of rejected requests per slot and the mean lifetime of instances. The number of rejected requests in each slot is divided by multiplication of the mean lifetime of VMs and the size of the time slots. Using the above framework, the provider adaptively updates the required parameters by the admission control algorithm.

At the beginning of each time slot, the predicted future demands and the computed pmf of u_t are fed into the admission control module that calculates the maximum number of reservations (r_0) that must be admitted in this time slot based on the admission control algorithm. The admission control module accepts reservation contracts while received demand is lower than r_0 during the time slot. Note that admission control algorithm is repeated for each time slot and only r_0 is used to perform actions during upcoming time slot. The produced result by the admission control algorithm remains valid as long as the observed demand is lower than the predicted demand or $r_0 < d_0^r$ for the current slot. The algorithm in admission controller module runs periodically and is executed at the beginning of each time slot. It then uses the updated information from the reserved capacity analyzer and prediction modules.

5.6 Performance Evaluation

In this section, we conduct two different groups of experiments to evaluate the proposed model. First, using a large scale simulation scenario, we evaluate the revenue management framework exploiting the proposed admission control algorithms. Then, we further evaluate the performance of the algorithms by conducting small scale simulations.

5.6.1 Framework Evaluation

Workload Setup

To our knowledge, no publicly available workload traces of real-world IaaS clouds currently exist as such information is often regarded by providers as being strictly confidential. Recently, Google has published a dataset pertaining to workloads on Google Compute Clusters [98]. This dataset includes the resource requirements of tasks submitted by users to a cluster of 12k physical machines over a time period of 29 days. Although the Google cluster does not constitute an actual public IaaS cloud, we argue that its usage can represent demands of public cloud users to some extent as it has resulted from the execution of actual cloud application services provided by Google.

An issue with these traces however is that they do not include any details on VM instances used to execute the application-level requests. For our experimentation, we therefore generate VM requests for each user as if the user was running the trace's workload in a virtualized IaaS cloud such as EC2. In this regard, it is worth mentioning that in the Google cluster tasks of different users might be scheduled onto a single machine, while in a public IaaS cloud a customer's VM executes only requests originating from applications that are hosted by that customer. In the following, we provide the details of the VM scheduling algorithm that is used to generate VM requests based on the workload of each user.

VM scheduling: The Google cluster trace includes records of a user/application submitting several *tasks*, each of which has resource requirements related to CPU, memory and disk [98]. We assume that the cluster's nodes are homogeneous as most of machines in the Google cluster have the same computing capacity, with 93% having the same computing capability [129].

We set our VM instance (i.e., capacity unit block) to have the same computing capacity as a node in the cluster. This enables us to accurately map resource requirements of tasks in the trace to VM instances. For each user, we use the following simple scheduling algorithm to instantiate and terminate VM instances based on the resource requirements of the tasks. Whenever, a user submits a task, the scheduling algorithm checks if there is available capacity in the pool of currently running VM instances; otherwise it instantiates a new VM instance. The scheduling algorithm also terminates running VM instances when there is no running task on the VM. The scheduling algorithm groups the VM requests that are instantiated at the same time as a single request for multiple VMs directed to the provider. As such, we obtain VM requests for each user and create a trace of $250,171$ VM requests.

Labeling Requests with Different Pricing Plans: After generation of the VM requests, they need to be assigned to one of the pricing plans offered by the provider. In IaaS public clouds, customers submit requests to a given market based on their applications' requirements and cost considerations. Customers who are interested to run their application at very low compute prices and those that require a large amount of capacity for a short period of time usually use spot market. The average lifetime of VM instances in this market is therefore shorter than in the other markets as instances face interruption from time to time. Applications with steady state or predictable long term usage usually utilize reserved instances. Their lifetime is therefore longer than in the other two markets. Applications with short term, spiky, or unpredictable workloads that cannot tolerate interruption usually rely on on-demand instances, which have a lifetime in between the other two categories.

On the basis of the above discussion, we use the following, necessarily synthetic, approach to associate each request to one of the markets. First, we normalize the lifetime of VM requests to the maximum lifetime in the traces and sort requests in ascending order of lifetime. Next, we generate random requests based on the three Gaussian distributions shown in Fig. 5.4. This results in $17,000$ requests assigned to the reserved market, $120,000$

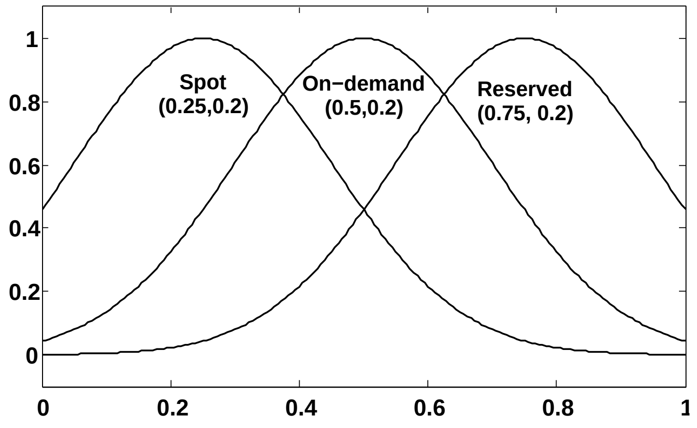


Figure 5.4: Three Gaussian functions for different pricing plans

requests to the spot market and all remaining requests to the on-demand market.

Reservation Requests: Up to this point, we generated workload traces for on-demand pay-as-you-go, reserved and spot instances. In order to generate requests for obtaining an actual reserved contract (reservation), we devise an online lazy reservation strategy for each user. Whenever the user submits a request directed to a reserved instance and does not have enough reserved capacity to handle the request, a new reservation contract is created. Using this technique, we assure that there is enough reserved capacity at each point in time to run all reserved instances of the user. If more than one reservation contract must be submitted to the system at the same time, we group them as a single reservation contract for multiple instances.

Simulation Setup

We extend CloudSim [16] to evaluate our revenue management framework. CloudSim is a discrete-event Cloud simulator that includes models of virtualized computing infrastructures and various VM provisioning policies. Our extensions relate to the support for the different pricing plans discussed in this chapter and the proposed revenue management system.

Pricing: We adopt the pricing details of Amazon EC2 in the us-east region at the time of writing. The VM configuration used for evaluating the revenue management system is aligned with Amazon EC2 *standard small instances*. Rates of \$0.06, \$0.021 and

\$0.012 per hour are used for the on-demand, reserved, and spot instances respectively and accordingly values of $\alpha \simeq= 0.35$ and $\beta = 0.2$ are computed. Similar to Amazon EC2, spot instances are not charged for their last partial hour upon their termination. On-demand or reserved instances that are terminated by their owner are charged for a discrete number of hours, with the last partial hour of usage accounted for as a full hour.

Since the used Google traces only span 29 days, we map each 5 minutes of workload data to one hour by linear scaling, resulting in a total simulation time of 12 months. We assume each reservation is effective for two months ($\tau = 60$ days) and that the upfront reservation fee is \$22.849 which is proportional to Amazon EC2's value of φ for a standard small instance (Linux, us-east, medium utilization) for a 1-year term.

Benchmark Algorithm: We compare the proposed *pseudo optimal* and *heuristic* algorithms with a benchmark algorithm that uses no admission control referred to as *no-control*. As its name implies, it admits all reservation contracts and gives preference to them over requests from the on-demand and spot markets. All reported revenues in Section 5.6.1 are normalized to the outcome of the *no-control* algorithm.

Experimental Results

We evaluate the revenue performance of the proposed framework using the *pseudo optimal* and *heuristic* algorithms. We consider the case where the workload is fixed throughout the simulation and the capacity (C) varies from 600 to 3400 with a step size of 400. We configure $B = 100$, $T = 75$ and $|Z| = 5$. The first and last two months of the 12-month simulation period are used as warm-up and cool-down periods, their respective outcomes are omitted from the experiment data. The lifetime of the on-demand instances in our workload trace does not precisely follow a geometric distribution. However, we assume the mean lifetime of on-demand instances to be equal to the expected value of the geometric distribution, i.e., $1/q$. We therefore set q to T divided by mean lifetime of on-demand instances in the workload.

The box plots in Fig. 5.5 show the revenue normalized to the *no-control* algorithm for 30 runs of the experiment. As expected, the revenue management system significantly improves revenue, especially when resources are scarce. As capacity increases, revenue gains decrease due to the fact that less opportunities for admission control arise and there is no resource contention. However, in no condition, it does lead to lower revenues compared to the *no-control* policy. In $C = 3400$ when the demand to supply ratio (DSR) is sufficiently low and there is no resource contention, both algorithms generate the same revenue performance as *no-control*. Note that, at a capacity level of 600 with a correspondingly high DSR, the *no-control* algorithm assigns the whole capacity to reservation contracts and all underutilized reserved capacity to the spot market. Our admission control algorithms increase revenue drastically under such high levels of resource contention. In such cases however, a real-world provider would likely increase C instead of entirely relying on admission control. This is an investment decision we would like to address in future developments of our revenue management framework.

According to Fig. 5.5, the *pseudo optimal* algorithm generates slightly more revenue than the heuristic algorithm; however, as stated before it suffers from a high order of computational complexity. The heuristic algorithm generates competitively high revenue with a significantly lower order of complexity. Therefore, in online cases, it can preform in a reasonable time frame with considerably finer values of T and B.

5.6.2 Evaluation of the proposed heuristic algorithms

In the previous section, we showed that the revenue management system performs well even in case of simple future demand prediction model and large values of T and B. Due to high computational complexity of the optimal algorithm, we could not compare our proposed algorithms with the optimal algorithm. In addition, prediction model errors and specific characteristics of the workload do not allow us to conduct fair experiments to show how close the algorithms can approximate the optimal solution. In this section, we evaluate the efficiency of the algorithms in comparison with optimal solution and we also investigate the impact of system parameters on the performance of the proposed algorithms in scenarios of smaller scale.

In our small scale evaluation, both capacity (C) and reservation period (τ) are set to 30. All pricing values are borrowed from the previous section and remain the same, except for the reservation fee which is updated to the 30-hour period, i.e., \$0.48. The

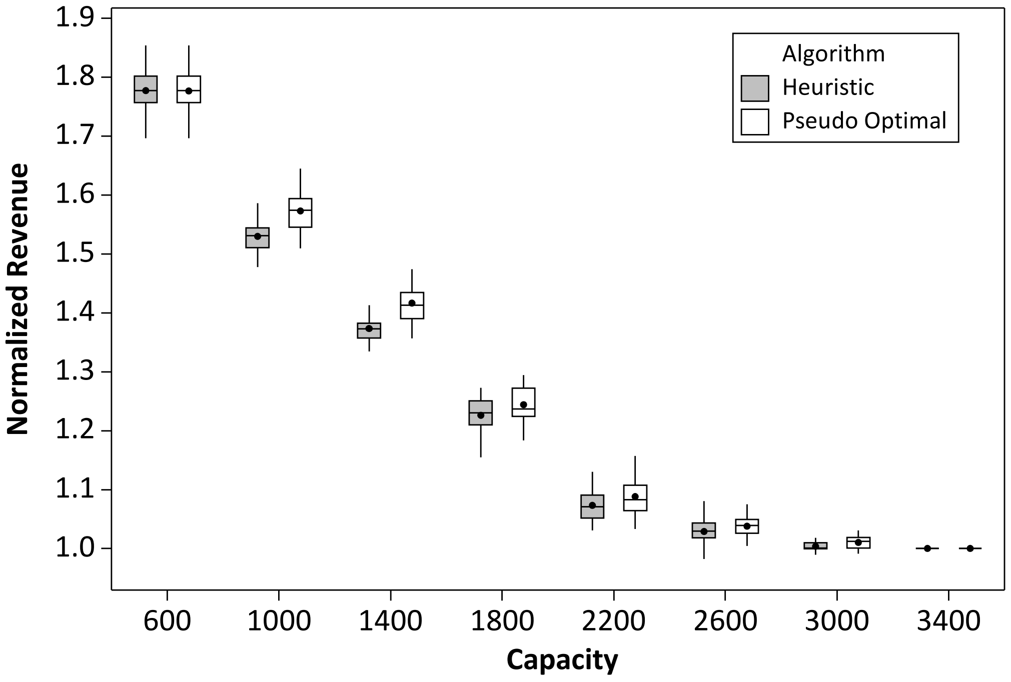


Figure 5.5: The revenue performance of the proposed revenue management framework under different algorithms normalized to the outcome of no admission control algorithm ($B = 100$ and $T = 75$).

amount of requests in the different markets are generated based on a Poisson distribution with parameter $\lambda = 1.5$ requests per hour. We used a Binomial distribution with parameters $q = 0.5$ and $n = |Z| = 5$ for the categorical pmf related to the reserved capacity utilization. All reported values of the revenue in this section are normalized to the outcome of the optimal algorithm and each experiment is carried out 30 times. For each experiment, we generate requests randomly according to the corresponding probability distributions. Afterwards, we schedule the arriving requests for a period of τ based on the computed actions by each algorithm separately. In this step, the expected revenue is computed based on the application of the same computed actions for 1000 runs in each of which the lifetime of on-demand pay-as-you-go requests are randomly generated based on the Binomial distribution and the related parameter q.

Fig. 5.6 shows box plots of the normalized revenue for the pseudo optimal and heuristic algorithms with different values of B and T when $q = 0.2$. The figure shows as T and B increase, the revenue performance of the algorithms decrease. The top left panel demonstrates head-to-head comparison of the revenue performance of the heuristic and pseudo

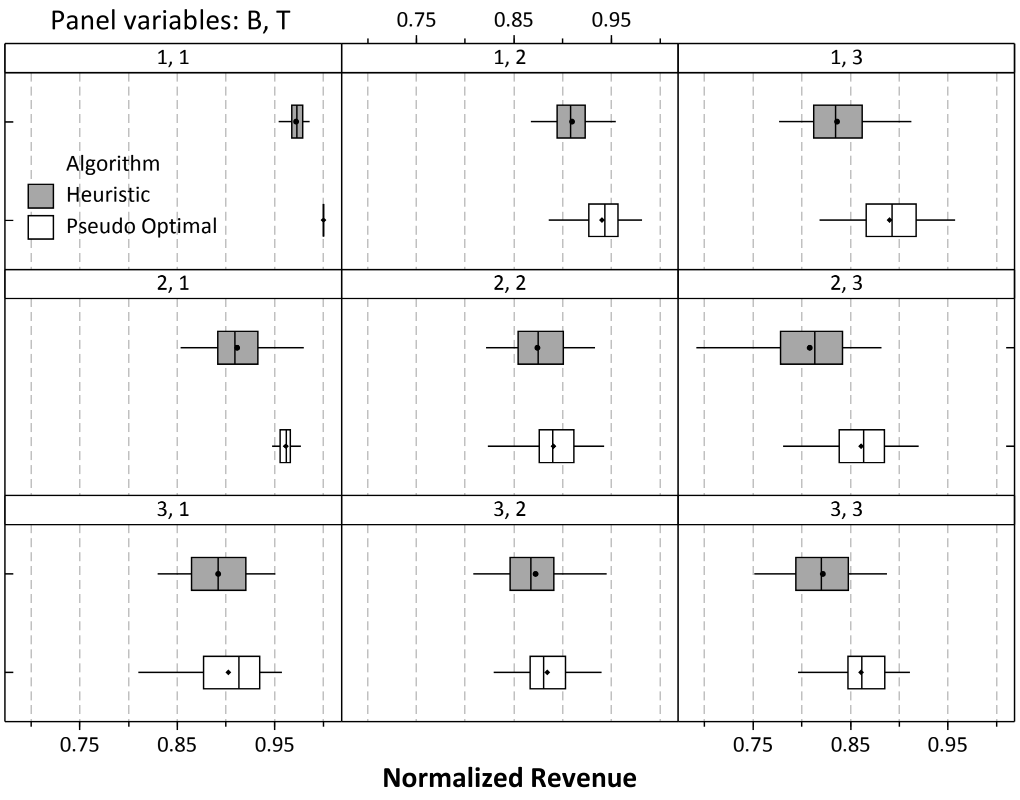


Figure 5.6: The revenue performance of the pseudo optimal and heuristic algorithms with different values of B and T. All values are normalized to the outcome of the optimal solution ($q = 0.2$).

optimal algorithm with $T = 1$ and $B = 1$ (i.e., optimal solution).

One important observation which may not be obvious from the figure is that even though the increase in the values B and T decreases the performance, the decrease is smaller when the two values are increased simultaneously. The reason is that scaling in only one dimension without considering the others causes the *rounding* errors to increase. In other words, increase in T enlarges the number of requests in one time slot and dividing large values to a predefined value of B results in a smaller rounding error.

Our sensitivity analysis reveals the only parameter which has significant effect on the revenue performance of the algorithms is q. Fig. 5.7 shows the box plots for the revenue performance of the heuristic algorithm with regards to q. As shown in the figure, as q increases, the revenue performance of the heuristic algorithm improves compared to the optimal solution. This is due to the fact that the optimal solution takes the probability

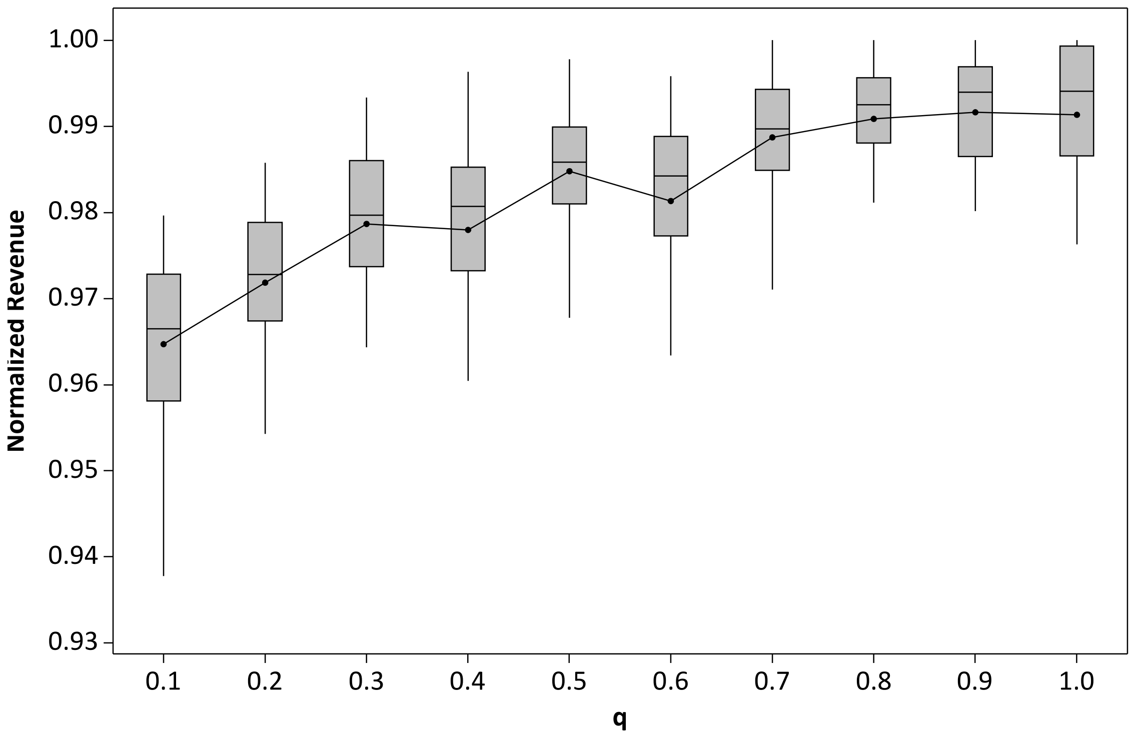


Figure 5.7: Impact of q, the termination probability of the running on-demand pay-as-you-go instance in the next time slot, on the revenue performance of the heuristic algorithm with $B = 1$ and $T = 1$. All values are normalized to the outcome of the optimal solution.

distribution of the instances' lifetime into account, while heuristic algorithm only uses the mean lifetime value to maximize revenue. Larger values of q result in smaller lifetime values, consequently the estimated shape of load generated by heuristic algorithm gets closer to the real shape of the load when eventually it perfectly matches at $q = 1$. This leads to a lower error for the solution found by the heuristic algorithm, around one percent at $q = 1$. Finally, it is worth mentioning that the low computational complexity and considerably high revenue performance of the heuristic algorithm make it a suitable choice by cloud providers aimed at revenue maximization in practical online cases.

5.7 Summary and Conclusion

In this chapter, we presented a revenue management framework to tackle the problem of optimal capacity control for allocating resources among customers of the IaaS cloud provider who are segmented into different cloud markets, i.e., reservation, on-demand pay-as-you and spot markets. The main challenge is that the provider must find an optimal capacity to admit demands from reservation market such that the expected revenue is maximized. We consider the stochastic lifetime of on-demand pay-as-you-go requests and reserved capacity utilization and we formulate the problem as a finite horizon Markov decision process. Finding the optimal solution is computationally prohibitive in practical settings, we therefore present two algorithms namely *pseudo optimal* and *heuristic* which reduce the computational complexity. Large-scale simulations driven by Google cluster-usage traces under Amazon EC2 pricing is conducted to evaluate the revenue performance of the proposed revenue management framework using our capacity control algorithms. We further evaluated the proposed algorithms with comparison to the optimal algorithm in a small scale scenario. Our experimental results suggest that significant revenue increment can be expected from using the proposed revenue management system given that sufficient resource contention is present in the system.

Chapter 6

An Auction Mechanism for a Cloud Spot Market

Dynamic resource pricing has recently been introduced by Infrastructure as a Service (IaaS) providers, in order to maximize profit and balance the supply and demand of resources. The design of a mechanism that efficiently prices perishable cloud resources in line with a provider's profit maximization goal, however, remains an open research challenge. In this chapter, we propose an adaptation of the Consensus Revenue Estimate auction mechanism to the setting of a multi-unit online auction for cloud resources. The mechanism is envy-free, has a high probability of being truthful, and generates a near optimal profit for the provider. We combine the proposed auction design with a scheme for dynamically calculating reserve prices based on data center Power Usage Effectiveness (PUE) and electricity costs. Our simulation-based evaluation of the mechanism demonstrates its effectiveness under a broad variety of market conditions. In particular, we show how it can improve on the classical uniform price auction and investigate the value of prior knowledge on the execution time of virtual machines, for maximizing profit.

6.1 Introduction

THE increased adoption and maturity of cloud computing offerings has been accompanied by a growing role and significance of pricing mechanisms for trading computational resources. Especially Infrastructure as a Service (IaaS) cloud providers that offer computational services in the form of Virtual Machine (VM) instances with specific resource characteristics, have gradually expanded their pricing plans in order to maximize their profits and further attract demand. Currently, the most widely used model remains a fixed *pay-as-you-go* pricing plan wherein the consumer is charged the amount of time a VM instance was used at a fixed rate. However, the fact that computational resources sold by a cloud provider can be characterized as a *non-storable* or *perishable* commodity[1], combined with the fact that demand for computational resources is non-uniform over time, motivates the use of dynamic forms of pricing in order to optimize revenue [54]. Through price adjustment based on actual (and possibly forecasted) supply and demand conditions, consumers can be incentivized to acquire spare capacity or shift demand from on-peak to off-peak hours. Consequently, both profit and consumer satisfaction can be increased.

Market-based pricing mechanisms that solicit reports (*bids*) from consumers and subsequently use an *allocation rule* and *pricing rule* to compute the allocation of resources to consumers and their associated prices respectively, are well fit to realize such dynamic forms of pricing. Recently, they have received significant attention for selling underutilized capacity in cloud infrastructures [61]. Well-designed auction mechanisms can be particularly effective since they: 1) incentivize users to bid in a truthful manner (i.e., report the price they are willing to pay for resources), 2) ensure resources are allocated to those who value them the most, and 3) correctly price resources in line with supply and demand conditions by creating competition among buyers.

Amazon Web Services (AWS) has adopted an auction-like approach to expand its pricing plans with Spot Instances for the Amazon Elastic Compute Cloud (EC2). In this scheme, consumers communicate their bids for a VM instance hour to AWS. Subsequently, AWS reports a market-wide *spot price* at which VM instance use is charged, while terminating any instances that are executing under a bid price that is lower than the market price. Although Amazon is not the only provider to offer dynamic pricing, it is currently the only IaaS provider that publicly offers an auction-like mechanism for selling IaaS resources. Nevertheless, attempts for creating such mechanisms have already been reported by other companies [115] and have also received attention by academia [9,25,127,139].

AWS has revealed no detailed information regarding their auction mechanism and the calculation of the spot price. At present, the design of an efficient, fair, and profit-maximizing auction mechanism for pricing cloud computing resources is an open re-

[1]Note that resources tied to a VM are qualified as non-storable (perishable), as a non-used hour of CPU time or memory space can never be reclaimed and therefore wastes data center capacity.

search challenge, and of great interest to cloud providers.

In this chapter, we design such an auction mechanism aimed at generating additional profit from the spare capacity of non-storable resources available in cloud data centers. We refer to the marketplace in which this mechanism is used to sell VMs as the cloud *spot market* (Fig. 6.1).

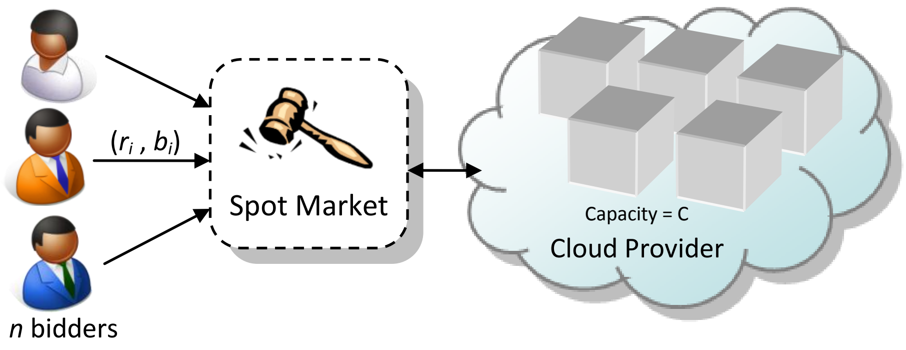


Figure 6.1: Spot market and auction mechanism

The spare capacity that can be offered by an IaaS cloud provider in the spot market is usually much larger than the demand[2]. Therefore a provider is potentially able to accept all consumer requests. In this context, popular auction mechanisms such as the second-price Vickrey [122] auction may fail to generate a reasonable revenue for the provider. In general, when supply exceeds demand, bidders are less motivated to bid competitively, which can prevent providers to collect an optimal revenue. Providers therefore require an auction mechanism that can maximize revenue while incentivizing bidders to reveal their true value. Hence, we restrict our focus to *truthful* auction designs. An auction mechanism is truthful if for each bidder i and any choice of order values by all other bidders, bidder i's *dominant strategy* is to report her private information with respect to her order truthfully. A strategy is dominant if a bidder cannot increase the pay-off derived from participating in the mechanism, by diverging from it.

If perfect knowledge about the distribution from which the bidders valuations are drawn is available, such a truthful auction mechanism can be designed [127]. Unfortunately, this is not always the case and pricing depends heavily on the accuracy of the

[2]This can be explained by the promise of Clouds providing infinite capacity of resources [7] and recent reports that suggest the overall utilization in large data centers is lower than 30% most of the time [36].

underlying market analysis. Such analysis also needs to be updated frequently in order to adapt to changes in the market. Moreover, since customers of cloud services are distributed globally and experience different latency for the same service, assuming that the valuations for all bidders are drawn i.i.d. might be invalid.

This chapter focuses on designing a truthful auction mechanism for a cloud spot market aimed at maximizing the cloud provider's profit. The cloud spot market context influences our auction design in the sense that the design needs to: support multi-unit bids, operate in an online recurrent manner, result in a single market-wide price and fair outcomes, operate under a limitation of the maximal quantity a consumer can request, operate without prior knowledge on the distribution of bidders' valuations, and finally, allow for reserve prices to be set during oversupply conditions. The chapter's key contributions are:

- The design and application of a multi-unit, online recurrent auction mechanism within the context of IaaS resource trading. The mechanism extends the off-line single-round auction with a single-unit demand model of the consensus revenue estimate (CORE) mechanism proposed by Goldberg and Hartline [37], to a two-dimensional bid domain. The proposed auction mechanism is envy-free, truthful with high probability and generates near optimal profit for the provider. It adopts a greedy approach for maximizing provider profits in the online setting. It is initially designed for the unlimited supply case, and is subsequently extended to the limited supply case.

- The evaluation of the proposed mechanism with respect to revenue generation, truthfulness, and bid rejection rates. Extensive simulation results are presented that demonstrate that it achieves near optimality w.r.t. maximizing revenue without requiring prior knowledge on the order distributions. It is also shown to achieve low bid rejection rates, mitigating the *bidder drop problem* in online mechanisms [54]. We compare the proposed mechanism to a clairvoyant and non-clairvoyant variant of the Optimal Single Price Auction and to the Uniform Price Auction.

- A clairvoyant optimal auction mechanism (HTA-OPT) that uses dynamic programming to calculate the set of accepted bids. HTA-OPT serves as a benchmark that is used to quantify the efficiency loss caused by the lack of information on the amount of time a bidder wants to run a VM, when applying the allocation rule in a single auction round.

- The presentation of a method for dynamically computing a *reserve price*, based on a coarse grained data center power usage model that can be used by the provider within the proposed auction mechanism. The resulting prices are shown to correspond to actual minimal spot prices observed on the EC2 spot market.

The remainder of this chapter is organized as follows: After reviewing related work in Section 6.2, we introduce required terminology and notations in Section 6.3. Sections 6.4, 6.5 and 6.6 discuss respectively the competitiveness, truthfulness and envy-freeness properties for our auction design. Section 6.7 describes the proposed auction mechanism, while Section 6.8 focuses on the limited supply setting and the computation of the reserve price in that setting. Section 6.9 describes the online version of the proposed auction mechanism and mechanisms used in the comparative analysis. Our experimental evaluation of the mechanism can be found in Section 6.10. We compare its performance to the Optimal Single Price Auction and the Uniform Price Auction, and investigate the impact of perfect knowledge on the execution time of a VM. We also provide simulation results concerning the probability that any bidder can benefit from an untruthful reporting of the number of VM instances required. Our conclusions follow in Section 6.11.

6.2 Related Work

The use of an auction-like mechanism to sell spare capacity in cloud data centers was pioneered in late 2009 by Amazon. In Amazon's spot market, customers bid the maximum hourly price they are willing to pay to obtain a VM instance[3]. All instances incur a uniform charge, the *spot market price*. According to Amazon, this price is set dynamically based on the relationship of supply and demand over time. A unique feature of

[3]`http://aws.amazon.com/ec2/spot-instances/`

spot instances is that the provider has the right to terminate them when their associated bid falls below the spot market price. As a result, the resulting quality of service (QoS) may be lower compared to *on-demand* and *reserved* instances, depending on the bid made. Current spot market data shows customers can acquire VMs at price reductions between 50% to 93% compared to on-demand instances.

Amazon has revealed little information on the pricing and allocation rules of their pricing mechanism. Ben-Yehuda et al. [9] examined the price history of the EC2 spot market through a reverse engineering process, and found that the mechanism was not completely driven by demand and supply. Their analysis suggests that spot prices are usually drawn from a tight, fixed price interval, and reflect a random non-disclosed reserve price. In this chapter, we propose an auction mechanism with transparent allocation and pricing rules, while sharing similar properties with the EC2 spot market.

Several authors have presented strategies for customers to utilize Amazon spot instances (cost-)effectively [22, 47, 114, 124, 135]. However, as of yet a limited amount of work has been conducted that focuses on the design of auction mechanisms to the benefit of cloud providers, and the associated algorithms for allocating resources and capacity planning to maximize the provider's revenue. The problem of dynamically allocating resources to different spot markets in order to maximize a cloud provider's revenue has been investigated by Zhang et al. [139]. Danak and Manno [25] present a uniform-price auction for resource allocation that suits the dynamic nature of grid systems. Mihailescu and Teo [69] investigate Amazon EC2's spot market as a case in a federated cloud environment. They argue that spot pricing used by Amazon is truthful only in a market with a single provider, and show that rational users can increase their utility by being untruthful in a federated cloud environment. Recently, Zaman et al. have investigated the applicability of combinatorial auction mechanisms for allocation and pricing of VM instances in cloud computing [137].

Wang et al. [127] proposed an optimal recurrent auction for a spot market based on the seminal work of Myerson [77]. The mechanism was designed in the context of optimally segmenting the provider's data center capacity between on-demand and spot market requests. Their work differs from ours since they adopt a Bayesian approach wherein it is assumed that the customers' private values are drawn from a known distribution. They also propose a truthful dynamic auction [128] that periodically computes the number of instances to be auctioned off in order to maximize providers revenue. Unlike EC2 spot marketplace, their approach offers guaranteed services (i.e., instances are never be terminated by the provider) and constant price over time (i.e. as the price is set for the user, it remains constant as long as the user holds the instance). Their auction charges each user a different price and does not generate a market-wide single price. Moreover, their auction mechanism requires a priori known distribution of valuations and near future demand prediction.

In contrast, we propose an auction mechanism designed to maximize profit based on a competitive auctioning framework proposed by Goldberg and Hartline [39]. The mechanism computes a uniform price outcome, and focuses on maximizing profit when the seller knows very little about the bidders valuations. In order to achieve truthfulness in this context, we rely on a *consensus estimation* technique [37].

Our work differs from that of Goldberg et al., in several aspects. First, their analysis relies on the assumption that each customer is restricted to formulate unit demand, which is not the case for cloud consumers as they can ask and bid for multiple VM instances. Consequently, we revisit the definition and truthfulness analysis of the mechanism for the multi-unit case. Second, their auction mechanism is designed for off-line single-round scenarios. The context of a cloud spot market however requires an online auction where customers arrive over time and resources allocated by VM instances can be released and subsequently reused by other consumers. We adopt a greedy approach in realizing the online character of the auction, and investigate its performance compared to a clairvoyant optimal mechanism that relies on dynamic programming. Finally, the production cost of goods is not taken into account in their work. In the IaaS setting, taking this cost into account is important as a seller has the option to either shut down server capacity or sell the capacity at a given *reserve price*. We add such reserve pricing to the mechanism and introduce a coarse-grained cost model to determine that.

Lee and Szymanski [54] have proposed an auction mechanism for time sensitive e-services where services must be resold for future time periods repeatedly. They investigated the *bidder drop problem* in recurrent auctions that occurs when the least wealthy bidders tend to withdraw from the future auction rounds due to repeatedly losing the auction. Our proposed auction is not specifically designed to address this issue, however our evaluation shows that it rejects a lower number of requests compared to the Optimal Single Price auction while generating near optimal revenue.

6.3 Preliminaries and Notation

Consider a cloud provider with capacity C for a specific type of VM. That is, at a given time t up to C instances of the specific type can be hosted simultaneously. The provider runs a sealed-bid auction, $\mathcal{A}$, to sell this capacity. First, we assume the case that the provider's capacity far exceeds the total demand, in line with the cloud's promise of delivering an unlimited supply of resources. Subsequently, we generalize the results to a scenario in which supply is limited and lower than total demand.

Suppose there are n customers joining the auction at time t. Each bidder i $(1 \leq i \leq n)$ requires q_i VM instances and has a private valuation v_i, denoting the maximum amount i is willing to pay for each VM instance per time slot (e.g., 1 hour). Customers submit an order (request), (r_i, b_i), where r_i represents the number of required VM instances and b_i the bid price. We denote by $\mathbf{d}$ the vector of all submitted orders. The ith element of $\mathbf{d}$, d_i, is the order by customer i.

Given $\mathbf{d}$, the provider (auctioneer) computes an allocation vector, $\mathbf{x} = (x_1, x_2, ..., x_n)$, and a price vector, $\mathbf{p} = (p_1, p_2, ..., p_n)$. The ith component x_i of the allocation vector indicates whether bidder i receives the r_i VMs requested in its order (if $x_i = 1$) or not ($x_i = 0$). A bidder for which $x_i = 1$ is called a *winner* and pays the corresponding price p_i, otherwise, the bidder is called a *loser* and does not make any payment to the mechanism. As we focus on single price auctions, all p_i are equal for all winning bidders and we therefore refer to the sale price as p. *Partial fulfillment* of requests, in which only a fraction of the number of VM instances requested is allocated to a winning bidder, is only considered in the case of limited supply and when $b_i = p$. We allow for partial fulfillment for those orders in line with the behavior of the EC2 spot market.

Note that bidders are individually rational users that try to maximize their utility. Therefore, as long as it is deemed beneficial, a customer will strategically misreport her bid or the required number of VMs i.e., $b_i \neq v_i$ or $r_i \neq q_i$, where v_i and q_i are private information known only to customer i. We define customer i's utility at time t for one time slot of VM usage as follows:

$$u_i(r_i, b_i) = \begin{cases} (q_i v_i - r_i p_i)x_i\,, & \text{if } b_i \geq p_i \text{ and } r_i \geq q_i; \\ 0\,, & \text{otherwise.} \end{cases} \tag{6.1}$$

The values of r_i and v_i for each customer are drawn from distributions that are unknown to the provider. Customer i's optimal bidding strategy must be defined so that it maximizes i's utility over all time slots. However, assuming that customers are not aware of the future and have no time-dependent valuation for resources, we define the utility function in (6.1) based on a single time slot. Winners in an auction round are awarded their requested VM instances and automatically attend the next round of the auction until they cancel their requests on their own account or they lose the auction. In the latter case, VMs held by an outbid customer are terminated by the provider without any prior notice.

The *holding time* of a VM is the specific amount of time a customer wants to run the VM. The VM's actual holding time might be smaller than the expected time if it is terminated by the provider instead of the owner. The holding time of a VM by the customer is not known to the provider (or to the mechanism) in advance. Therefore, in our model, a provider acts in a greedy manner to maximize revenue according to the arriving requests and the current existing requests in each round of auction. This can be modeled as a single round auction which is recurrently conducted by the provider as new requests arrive or current requests are terminated. In section 6.10.4, we compare the performance of this greedy strategy to the optimal strategy that has prior knowledge on the VM holding time. From this point onwards, we limit our discussion only to a single round of the auction. In Section 6.9, we introduce the recurrent version of the mechanism.

6.4 Competitive Framework

The revenue generated by auction $\mathcal{A}$ in a time slot equals:

$$\mathcal{A}(\mathbf{d}) = \sum_i r_i p_i \,. \tag{6.2}$$

The problem of maximizing revenue in an auction for cloud resources can be solved optimally if the seller knows the distribution from which the bidders' valuations are drawn i.i.d. [127]. In conventional economics this is called *Bayesian Optimal Mechanism Design* [77,80]. However, we assume that the distributions from which the bidder's private information are drawn are unknown to the provider. Therefore, we base our approach on the competitive mechanism design proposed by Goldberg et al. [39]. We will compare the revenue attained by our mechanism to that of the *Optimal Single Price auction* for the unlimited capacity case.

Definition 1. *The* Optimal Single Price auction, *$\mathcal{F}$, is defined as follows: Let $\mathbf{d}$ be an order vector. Without loss of generality, suppose the components of $\mathbf{d}$ are sorted in descending order by bid values. So, (r_i, b_i) is the ith largest bid in $\mathbf{d}$ regardless of r_i. The auction $\mathcal{F}$ on input $\mathbf{d}$ determines the value k such that $b_k \sum_{i=1}^{k} r_i$ is maximized. We denote by $\sigma_k(\mathbf{d})$ the sum of the number of requested instances in the sorted vector of orders from the first order to kth order ($\sigma_k(\mathbf{d}) = \sum_{i=1}^{k} r_i$). All bidders with $b_i \geq b_k$ win at price b_k and all remaining bidders lose. Thus, the revenue of $\mathcal{F}$ on input $\mathbf{d}$ is*

$$\mathcal{F}(\mathbf{d}) = \max_i \; b_i \sigma_i(\mathbf{d}) \,. \tag{6.3}$$

If more than one value of i maximizes $b_i\sigma_i(\mathbf{d})$, choosing the price point that results in a lower transacted volume is preferable considering the cost of accommodating VM instances (e.g., electricity cost). From this point forward, we assume $\mathbf{d}$ is sorted decreasingly by bids values (b_i), unless otherwise mentioned.

We are interested in an auction mechanism that is competitive with $\mathcal{F}$ on every possible input; however, if a single bidder's utility dominates the total utility of the other bidders, no auction can compete with $\mathcal{F}$ as shown by Goldberg et al. [39]. We do not consider this to be an issue in our setting, because the cloud environment can be viewed as a mass-market where the number of winners of the optimal single price auction is typically large. In a mass-market, removing one order does consequently not change the maximum extractable profit significantly.

Definition 2. (Mass-market)*: Let $\mathcal{F}(\mathbf{d})$ be the revenue of $\mathcal{F}$ and $h_b(\mathbf{d})$ denote the maximum value of b in $\mathbf{d}$, then $\mathcal{F}(\mathbf{d}) \gg h_b(\mathbf{d})$ in mass-markets, which implies that $\mathcal{F}$ sells $m \gg 1$ units.*

We say that auction $\mathcal{A}$ is competitive if there exists a constant β such that $\mathcal{A}(\mathbf{d}) \geq \mathcal{F}(\mathbf{d})/\beta$. For a randomized mechanism[4], the previous equation for competitiveness becomes:

$$\mathbf{E}[\mathcal{A}(\mathbf{d})] \geq \frac{\mathcal{F}(\mathbf{d})}{\beta}.$$

Assuming the fact that $\mathcal{F}$ sells at least m units, we define $\beta(m)$-competitiveness for a mass-market as below:

Definition 3. *Auction $\mathcal{A}$ is $\beta(m)$-competitive for a mass-market if for all order vectors* $\mathbf{d}$ *such that $\mathcal{F}$ sells at least m units, we have:*

$$\mathbf{E}[\mathcal{A}(\mathbf{d})] \geq \frac{\mathcal{F}(\mathbf{d})}{\beta(m)}. \tag{6.4}$$

6.5 Truthfulness

Let $\mathbf{d}_{-i}$ denote the vector of orders $\mathbf{d}$ with $(r_i,\ b_i)$ removed, i.e., $\mathbf{d}_{-i} = ((r_1, b_1), \ldots, (r_{i-1}, b_{i-1}), (r_{i+1}, b_{i+1}), \ldots, (r_n, b_n))$, and further introduce the notation $\mathcal{F}((r_i, b_i), \mathbf{d_{-i}}) = \mathcal{F}(\mathbf{d})$.

Proposition 1. *$\mathcal{F}$ is not truthful.*

Proof. Suppose $\mathcal{F}$ is truthful, then utility for each bidder i is maximized if $b_i = v_i$ and $r_i = q_i$ for any choice of $\mathbf{d}_{-i}$.

Consider $\mathbf{d}$ as any arbitrary vector of orders, assume $\mathcal{F}(\mathbf{d})$ is the maximum revenue by $\mathcal{F}$ and b_k is the sale price. Suppose $\mathcal{F}_2(\mathbf{d})$ is the second largest revenue which can be obtained by $\mathcal{F}$ and we limit $\mathbf{d}$ to those vectors such that $\mathcal{F}(\mathbf{d}) > \mathcal{F}_2(\mathbf{d})$. Given a fixed

[4]The mechanism's allocation and/or pricing rule procedure has a randomized component.

$\mathbf{d}_{-k}$, r_k, q_k, bidder k is able to reduce her bid from b_k to b'_k and still be the winner as long as $\mathcal{F}((r_k, b'_k), \mathbf{d}_{-k}) > \mathcal{F}_2(\mathbf{d})$. As a result, fixing other variables and considering that bidder k is a winner ($x_k = 1$), bidder k is able to increase her utility from $q_k v_k - r_k b_k$ to $q_k v_k - r_k b'_k$. So there exists a $\mathbf{d}_{-i}$ and a bidder i such that u_i can be increased by misreporting i's true value, i.e., $b_i \neq v_i$. This contradicts the supposition that $\mathcal{F}$ is truthful. A similar proof can be constructed for the number of requested instances, which we omit here for space considerations. □

In order to create a truthful auction, an intuitive idea is to design the mechanism in a way that a bidder believes that her own order does not affect the price she pays. This is called an order-independent auction since the price the bidder is offered in the auction is independent of the bidder's bid value [39]. An order-independent auction can be viewed as a function that maps $\mathbf{d}_{-i}$ to a price for each bidder.

Definition 4. *The* order-independent auction *offers a price p_i to bidder i computed by the function f according to the order vector $\mathbf{d}_{-i}$, i.e., $p_i = f(\mathbf{d}_{-i})$. If bidder i's bid is greater or equal to p_i ($b_i \geq p_i$), the bidder wins the auction ($x_i = 1$) and pays p_i; otherwise the bidder loses the auction ($x_i = 0$) and pays zero.*

Lemma 1. *The* order-independent auction *is truthful.*

Proof. Following Definition 4, bidder i's order does not affect the price she ends up paying, so the bidder is not able to increase her utility by changing her order. As a result, the bidder has no incentive to misreport her bid or quantity levels as this does not change the amount she pays. □

Following [39], we introduce the *optimal order-independent auction*. To define it, first we define the notion of the optimal single sale price for a set of orders.

Definition 5. *Let $\mathbf{d}$ be a sorted vector of orders by descending values of bids. Denote $opt(\mathbf{d})$ the optimal single sale price for $\mathbf{d}$ that maximizes the revenue for the auctioneer, i.e.,*

$$opt(\mathbf{d}) = \underset{b_i}{\operatorname{argmax}} \; b_i \sigma_i(\mathbf{d}) \,. \tag{6.5}$$

Now we can define the *optimal order-independent auction*, which is a truthful auction, as follows:

Definition 6. *The* optimal order-independent auction *is defined by the order-independent function f such that $f(\mathbf{d}_{-i}) = opt(\mathbf{d}_{-i})$.*

Unfortunately, even though the optimal order-independent auction is truthful, it has two main characteristics that make it unsuitable for our purposes. Firstly, it is not single price and secondly, a bidder j might lose the auction while bidder i with $b_i < b_j$ wins and is charged $p_i < b_j$. In this case the auction's outcome is not fair and the losing bidder envies the winning bidder's outcome. This might happen as the sale price for bidder i is computed based on $\mathbf{d}_{-i}$ which is different for each bidder. Proof of Lemma 2 provides examples of these outcomes.

6.6 Envy-freeness

In an *envy-free auction* no bidder can increase its utility by adopting another bidder's outcome. For our case, an envy-free auction requires a single sale price. All bidders willing to pay this price are provided with VM instances and charged at that price uniformly.

In this work, it will be irrelevant how bids that equal the sale price are treated, however, we assume that they are always provided with VM instances if the provider's capacity allows for it. Note that, according to the utility function in Equation 6.1, the utility value (u_i) is always zero for those bidders with true bid values (v_i) equal to p, irrespective of them winning or losing. Therefore, those bidders are assumed to have no preference over the two possible outcomes.

Lemma 2. *The* optimal order-independent auction *is not envy-free.*

Proof. It suffices to construct an example showing that the optimal order-independent auction is not single price. Consider three bidders with the following orders $d_1 = (1, \$8)$, $d_2 = (2, \$7)$, and $d_3 = (4, \$2)$. In order to calculate the sale price for each bidder i, first we obtain $\mathbf{d_{-i}}$ by removing bidder i's order from $\mathbf{d}$. Then $opt(d_{-i})$ is computed according to (6.5). Performing the above process for all bidders, we obtain the outcome for each bidder as follows. Bidder one and two win the auction and pay \$7 and \$2 respectively, while bidder three loses the auction and pays zero. This shows that optimal order-independent auction is not single price.

In addition, the order-independent auction is not fair as there are situations in which a bidder might lose the auction while another bidder with a lower bid price wins the auction. Consider four bidders with orders $d_1 = (2, \$13)$, $d_2 = (5, \$3)$, $d_3 = (1, \$2)$ and $d_4 = (20, \$1)$. Bidder one and three win the auction and both pay bidder four's bid price, i.e., \$1 per instance, while bidder two with a bid price higher than bidder three (\$3 $>$ \$2) loses the auction. $\square$

Goldberg and Hartline [38] showed that no truthful, envy-free auction can be constant competitive and they provided the lower bound of $log(n)/log \;\; (log(n))$ with n the number of bidders. In order to obtain a constant competitive auction mechanism, we relax the assumption of truthfulness and extend the proposed Consensus Revenue Estimate (CORE) auction [38] for our case. The proposed auction is envy-free but is only truthful with *high probability*.

Definition 7. *An auction is truthful with probability $1 - \epsilon$ if the probability that any bidder can benefit from an untruthful bid is at most ϵ. If ϵ is inverse polynomial in some specified parameters of the auction (such as the number of items or bidders) then we say the mechanism is* truthful with high probability.

In the following section, we show that the proposed auction mechanism is truthful with high probability with respect to the bid price dimension. We also provide simulation results concerning the probability that any bidder can benefit from an untruthful reporting of the number of VM instances required.

6.7 Extended Consensus Revenue Estimate Auction

Recall that the optimal order-independent auction in Section 6.5 is truthful since it is order-independent. Due to the fact that it is not single price, and therefore not envy-free, it is not suitable for our problem context. The question therefore arises as to how a single price can be computed for an order-independent auction while attaining the revenue of the optimal auction, that is, $\mathcal{F}(\mathbf{d})$. It is clear that $\mathcal{F}(\mathbf{d})$ cannot be computed from $\mathbf{d}_{-i}$ and consequently, a function f that generates the optimal sale price based on $\mathbf{d}_{-i}$ cannot be built. Therefore, we are interested in a mechanism that provides us with a sufficiently accurate estimate of $\mathcal{F}(\mathbf{d})$ that is constant on $\mathbf{d}_{-i}$ for all i (i.e., it achieves *consensus*). If $\mathcal{F}(\mathbf{d}_{-i})$ is limited by a constant fraction of $\mathcal{F}(\mathbf{d})$, it is possible to pick a good estimate of $\mathcal{F}(\mathbf{d})$ such that it achieves consensus with high probability [38]. In the remainder of this section, we will outline how this estimate is computed.

In mass-markets such as clouds, $\mathcal{F}(\mathbf{d})$ is much larger than the highest bid. Let $h_b(\mathbf{d})$ denote the maximum bid value in $\mathbf{d}$, then $\mathcal{F}(\mathbf{d}) \geq \alpha h_b(\mathbf{d})$ in mass-markets, which implies that $\mathcal{F}$ sells at least α units.

Let m ($m \geq \alpha$) be the number of sold units in $\mathcal{F}$. If m is sufficiently large and the maximum number of units that can be requested by a customer is limited, removing an order does not change $\mathcal{F}(\mathbf{d})$ considerably. We show this in Lemma 3.

Enforcing a restriction on the maximum number of VM instances that can be simultaneously acquired by a customer is reasonable and done by public cloud providers such as Amazon[5]. Such restriction reduces the chance of system stability being threatened by very large unpredicted requests. In addition, it reduces the risk of starvation for customers with small requests in the presence of wealthy customers.

Lemma 3. *Let r denote the supremum of the number of requested units in $\mathbf{d}$, i.e., $r_i \leq r$ for all bidders, $1 \leq i \leq n$. If m, the number of sold units in $\mathcal{F}$, is sufficiently large, then for any i,*

$$\frac{m-r}{m}\mathcal{F}(\mathbf{d}) \leq \mathcal{F}(\mathbf{d}_{-i}) \leq \mathcal{F}(\mathbf{d})\,. \tag{6.6}$$

Proof. Without loss of generality, suppose $\mathbf{d}$ is sorted in descending order of bids (b_i), i.e., $b_1 \geq b_2 \geq ... \geq b_n$. Suppose k is the rank of the bidder in $\mathbf{d}$ whose bid maximizes $b_i\sigma_i(\mathbf{d})$, i.e., $\mathcal{F}(\mathbf{d}) = b_k\sigma_k(\mathbf{d})$. By removing order i from $\mathbf{d}$, the maximum reduction in $\mathcal{F}(\mathbf{d})$ is r_ib_k (when $i \leq k$), and the minimum reduction is zero (when $i > k$). Therefore,

$$\mathcal{F}(\mathbf{d}) - r_ib_k \leq \mathcal{F}(\mathbf{d}_{-i}) \leq \mathcal{F}(\mathbf{d})\,.$$

[5] `http://aws.amazon.com/ec2/faqs/#How_many_Spot_Instances_can_I_request`

$$m = \sum_{j=1}^{k} r_j \Rightarrow b_k = \frac{\mathcal{F}(\mathbf{d})}{m},$$
$$r_i \leq r \Rightarrow r_i b_k \leq r \frac{\mathcal{F}(\mathbf{d})}{m} \Rightarrow$$
$$\frac{m-r}{m}\mathcal{F}(\mathbf{d}) \leq \mathcal{F}(\mathbf{d}_{-i}) \leq \mathcal{F}(\mathbf{d}).$$

□

We introduce ρ for $\frac{m}{m-r}$. In mass-markets, $\frac{1}{\rho}\mathcal{F}(\mathbf{d}) \leq \mathcal{F}(\mathbf{d}_{-i}) \leq \mathcal{F}(\mathbf{d})$, meaning that $\mathcal{F}(\mathbf{d}_{-i})$ is at least a constant fraction of $\mathcal{F}(\mathbf{d})$.

The Extended Consensus Revenue Estimate Auction (Ex-CORE) combines two general ideas as its name implies: *consensus estimation* and *revenue extraction*. For consensus estimation, it picks a function that estimates $\mathcal{F}(.)$ with high quality and achieves consensus with high probability. A function that works well in our case is g, defined as:

$g(\mathcal{F}(.)) = \mathcal{F}(.)$ rounded down to the nearest c^{l+u}

where $c > \rho$ is a constant chosen as to maximize the quality of the estimation, u is a uniform random value on $[0, 1]$, and l is the largest integer so that $c^{l+u} \leq \mathcal{F}(.)$.

Lemma 4. *[37] For $c > \rho$ and any $\mathbf{d}$ with $\frac{1}{\rho}\mathcal{F}(\mathbf{d}) \leq \mathcal{F}(\mathbf{d}_{-i}) \leq \mathcal{F}(\mathbf{d})$, the probability that g outputs a value which is constant on all $\mathbf{d}_{-i}$ (i.e., achieves consensus) is $1 - \log_c \rho$.*

Lemma 5. *[37] If* payoff *for g, γ_g, is defined as:*

$$\gamma_g(\mathcal{F}(.)) = \begin{cases} g(\mathcal{F}(.)), & \textit{if } g \textit{ achieves consensus;} \\ 0, & \textit{otherwise.} \end{cases} \tag{6.7}$$

then for all $\mathcal{F}(.)$, we have:

$$\mathbf{E}[\gamma_g(\mathcal{F}(.))] = \frac{\mathcal{F}(.)}{ln(c)}\left(\frac{1}{\rho} - \frac{1}{c}\right). \tag{6.8}$$

Let us now discuss how to choose the value of c. We are interested in the expected payoff to be large relative to $\mathcal{F}(.)$, i.e., $\mathbf{E}[\gamma_g(\mathcal{F}(.))]/\mathcal{F}(.)$ is large over different values of $\mathcal{F}(.)$. For a fixed value of ρ, we can choose the value of c that maximizes $\frac{1}{ln(c)}\left(\frac{1}{\rho} - \frac{1}{c}\right)$. This function is differentiable on $c \in (1, \infty)$ and it has an absolute maximum on that interval. Therefore, by taking the derivative of it w.r.t. c and setting it to zero, we have:

$$\frac{\partial \mathbf{E}[\gamma_g(\mathcal{F}(.))]/\mathcal{F}(.)}{\partial c} = 0 \Rightarrow$$
$$\frac{\rho\, ln(c) + \rho - c}{\rho\, c^2\, ln^2(c)} = 0, \ \ \rho > 1, \ \ c > \rho \Rightarrow$$
$$\rho\, ln(c) + \rho - c = 0 \tag{6.9}$$

Note that (6.9) does not have an exact solution and needs to be solved by numerical methods.

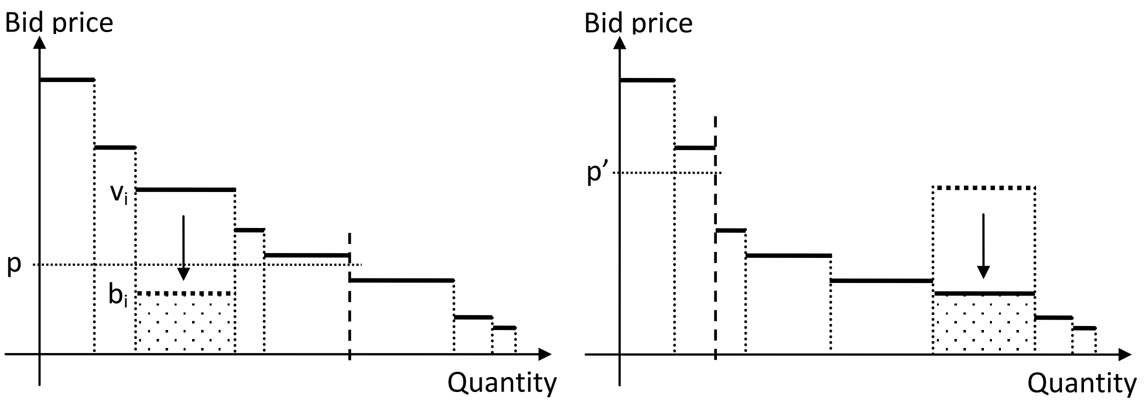
The second component of Ex-CORE, a revenue extraction mechanism, extracts a target revenue from the set of bidders if this is possible. The algorithm is based on the cost sharing mechanism proposed by Moulin and Shenkar [76]. Given an order vector $\mathbf{d}$ sorted in descending order of bids and a target revenue R, the revenue extractor function $e_R(\mathbf{d})$ finds the largest k such that $R/\sigma_k(\mathbf{d}) \geq b_k$. In other words, it finds the k bidders with the highest bid values that allow for the extraction of R. R is then shared among these k bidders based on the number of requested instances by each bidder, that is, each of these bidders are charged $R/\sigma_k(\mathbf{d})$. If no subset of bidders can share R, the auction has no winners.

Lemma 6. *Given a target revenue R, the revenue extraction mechanism is truthful for the price dimension but not for the quantity dimension.*

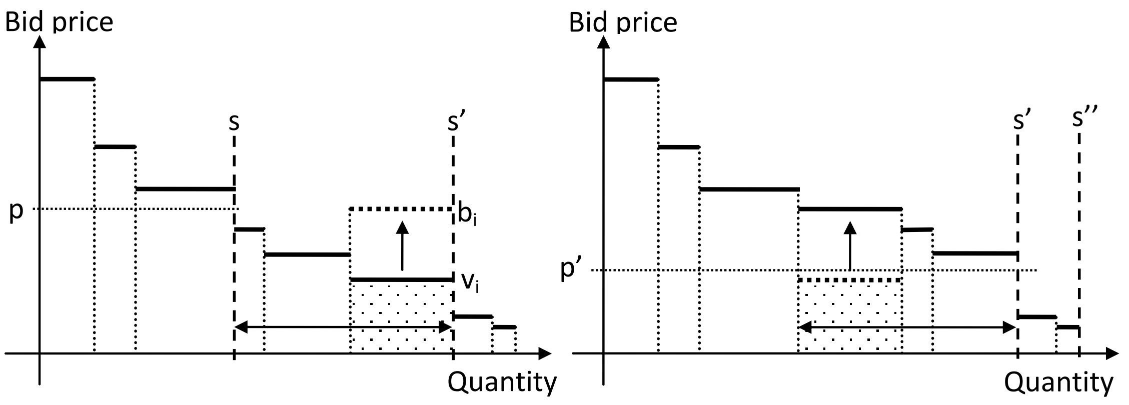
Proof. Without loss of generality, we consider $\mathbf{d}$ as sorted. The revenue extraction mechanism is truthful if $u_i(q_i, v_i) \geq u_i(r_i, b_i)$ for all values of b_i and r_i and for every bidder i, $1 \leq i \leq n$. First, we show that given a fixed r_i any untruthful submission of the bid price, i.e., $b_i \neq v_i$ decreases bidder's i utility. It suffices to consider the following two cases:

Case 1: Suppose the truthful submission ($v_i = b_i$) leads to bidder i winning the auction, it is easy to verify that reporting $b_i > v_i$ only decreases the rank of bidder i in $\mathbf{d}$, assuming $\mathbf{d}$ remains unchanged except for bidder i. Therefore, it does not change the sale price and as a result, bidder i's utility also remains unchanged.

If bidder i reports $b_i < v_i$, as long as $b_i \geq p$ (p is the sale price), p remains unchanged. Hence, bidder i's utility does not increase or decrease. However, as soon as $b_i < p$,



(a) Reporting $b_i < v_i$ increases the price to $p' > v_i$.



(b) Reporting $b_i > v_i$ decreases the price to $p' > v_i$.

Figure 6.2: Effect of misreporting true value on the sale price. Truthful submission leads to (a) winning and (b) losing.

bidder i loses the auction, the sale price rises, and the bidder's utility drops to zero. This is illustrated in Fig. 6.2a. Consequently, submitting $b_i < v_i$ might not improve bidder i's utility and might reduce it to zero.

Case 2: Suppose the truthful submission ($v_i = b_i$) leads to the bidder losing the auction, then reporting $b_i < v_i$ would clearly not change the zero utility of the bidder.

If reporting $b_i = v_i$ leads to bidder i losing the auction, it follows that $p > v_i$. Assume $p = R/s$, where s is the sum of the number of requested units by largest group of k bidders with highest bid values that can at least generate a revenue of R. Consider $s' = \sigma_i(\mathbf{d})$, as a result $s' > s$, since we know b_i is a losing bid.

Suppose bidder i reports her bid $b_i > v_i$, we argue that new sale price p' is always larger than v_i ($p' > v_i$). That is, increasing b_i might increase s up to s' at most. This is shown in Fig. 6.2b.

Using reductio ad absurdum, assume by increasing b_i, s can be increased to a value $s'' > s'$. Hence, we know that there is a bidder j whose bid price, b_j, is larger than R/s'' ($R/s'' \leq b_j$). We know that $i < j$ and $b_j \leq v_i$, because $s'' > s'$ requires j to be placed after i in the sorted vector. If $R \leq s''b_j$ after increasing b_i, then $R \leq s''b_j$ before increasing as well, because bidder i is placed in the lower rank either bidding at $b_i = v_i$ or $b_i > v_i$ in the sorted vector of orders. That is, bidder i is a winner in either of cases. This contradicts our initial assumption that reporting $b_i = v_i$ leads to bidder i losing the auction. So, $s'' \leq s' \Rightarrow p' > v_i$.

Hence, bidding $b_i > v_i$ leads to negative utility for bidder i and bidder i would be worse off.

Second, we provide an example that demonstrates that the revenue extraction mechanism is not truthful for the quantity dimension. That is, bidders are able to increase their utility by misreporting their required number of instances. Assume $R = \$7$ and an order vector $\mathbf{d} = \{(1, \$8), (5, \$1)\}$. Bidder one is charged $7/1 = 7$ and bidder two loses according to the revenue extraction mechanism. The utility for bidder one is then computed as follows: $1 \times 8 - 1 \times 7 = 1$. Now, consider that bidder one misreports 2 as the required number of instances. Then, the largest group of bidders able to share R includes the orders of both bidders. Therefore the price for bidder one is $7/7 = 1$ and its utility is computed as: $1 \times 8 - 2 \times 1 = 6$.

□

Definition 8. Extended Consensus Revenue Estimate Auction (Ex-CORE): *For constant c, and a random value u, uniformly chosen from $[0, 1]$, find $g(.)$ as $\mathcal{F}(.)$ rounded down to nearest c^{l+u} for integer l. The sale price by Ex-CORE is then defined as $p = e_R(\mathbf{d})$ where $R = g(\mathcal{F}(\mathbf{d}))$.*

Lemma 7. *For order vector $\mathbf{d}$, constant c and a choice of u, if $g(\mathcal{F}(\mathbf{d}_{-i})) = R$ for all i, $1 \leq i \leq n$, i.e., it is a consensus, then the Ex-CORE auction is truthful with respect to bid prices.*

Proof. It suffices to show that if $g(\mathcal{F}(\mathbf{d}_{-i})) = R$ for all i, no bidder can increase her utility by bidding any value other than their true bid value. Note that If $g(\mathcal{F}(\mathbf{d}_{-i})) = R$ for all

i then $g(\mathcal{F}(\mathbf{d})) = R$. Now consider that bidder i submits an order (r_i, b_i) where $b_i \neq v_i$ resulting in $\mathbf{d}'$ ($\mathbf{d}'$ is identical to $\mathbf{d}$ except for bidder i's bid price).

As long as $g(\mathcal{F}(\mathbf{d}')) = g(\mathcal{F}(\mathbf{d})) = R$, bidder i is not able to benefit out of misreporting. Because the sale price p is computed as $p = e_R(\mathbf{d})$, and according to Lemma 6, the revenue extraction mechanism is truthful. Therefore, bidder i's utility cannot be improved by misreporting v_i; thus bidder i's best strategy is to bid at v_i.

The proof is in fact very straightforward. For every user i, since $\mathcal{F}(\mathbf{d}_{-i}) = \mathcal{F}(\mathbf{d})$, changing bid b_i to b'_i will lead to a new order vector $\mathbf{d}'$ the same as the original $\mathbf{d}$ except component i. As a result, $\mathbf{d}'_{-i} = \mathbf{d}_{-i}$. Hence $g(\mathcal{F}(\mathbf{d}'_{-i})) = g(\mathcal{F}(\mathbf{d}_{-i})) = g(\mathcal{F}(\mathbf{d})) = R$. This essentially implies that user i will be given exactly the same price as before. Consequently, the sale price cannot be decreased and bidder i's utility cannot be increased.

□

Proposition 2. *The Extended Consensus Revenue Estimate Auction (Ex-CORE) is envy-free, truthful with probability* $1 - \log_c \rho$ *for the bid price dimension, and* $\frac{1}{ln(c)}\left(\frac{1}{\rho} - \frac{1}{c}\right)$*-competitive for mass markets.*

Proof. Definition 8 and Lemmas 4, 5 and 7 are enough to prove the proposition. □

6.7.1 Discussion

The Ex-CORE auction is not two-dimensionally truthful because the revenue extraction mechanism is not truthful for the quantity dimension (Lemma 6). In the cloud spot market however, no customer has an incentive to request fewer instances than needed (as $u_i(r_i, b_i) = 0$ whenever $r_i < q_i$). Our detailed investigation of the proposed mechanism shows that bidders are able to increase their utility in some cases by requesting a higher number of instances than what they actually require. Devising a two-dimensional truthful mechanism for this highly complex strategy space remains as a future work. Nevertheless, we believe that the proposed mechanism retains its practical value due to several key reasons.

First, users who misreport the required number of instances end up paying for a higher number of instances. In order to increase utility, the increment in r_i must cause a sufficient reduction in the market price to compensate for the surplus cost a bidder pays for the additional instances. Considering that the bidder is not aware of the other orders, there is always a risk of decrease in utility by misreporting.

Second, $\mathcal{F}(\mathbf{d})$ is monotonically increasing w.r.t r_i (the rationale is intuitive) and Ex-CORE calculates the sale price based on the estimation of $\mathcal{F}(\mathbf{d})$. Given that r (the maximum number of requested instances) is a constant and $m \to \infty$ in a cloud mass-market, in expectation R (the estimated value of $\mathcal{F}(\mathbf{d})$) rises as the bidder increases demand. The revenue extraction mechanism computes the price by $R/\sigma_k(\mathbf{d})$. Therefore the risk of increasing the market price increases by misreporting the number of required instances, as the numerator of the fraction (R) is increasing while there is no certainty about the decrease or increase in the denominator ($\sigma_k(\mathbf{d})$).

Last but not the least, r is constrained by a limit. As the number of sold instances in the cloud market is usually high, the effect on the market price of a bidder misreporting demand is typically low given the assumption of non-collusive behavior of bidders.

In Section 6.10, we demonstrate through simulation that in markets of sufficient size, an individual bidder indeed has a very low probability of gaining utility by misreporting VM demand.

6.8 Limited Supply and Reserve Price

Up to this point, we have considered an unlimited capacity setting. In reality, however, situations arise wherein a cloud provider needs to reject requests due to lack of supply. We modify the auction mechanisms to take into account that $C(t)$, the number of VM instances available for sale at time t, can be lower than the demand.

As the provider wishes to maximize revenue, it can select a set of high-value bidders such that the total amount of requested VMs by this set is smaller or equal to $C(t)$. This set of bidders subsequently participates in the auction mechanism for the unlimited supply case, while the remaining bids are rejected. Fig. 6.3 depicts how supply is limited by $C(t)$. This method allows us to extend our discussion into the bounded supply case. In order to be envy-free in the bounded supply case, we need to ensure that none of the bidders win

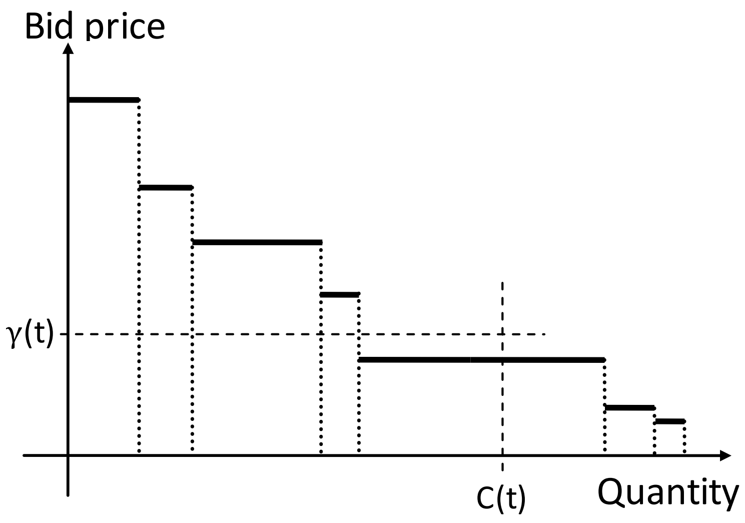


Figure 6.3: Supply limited by capacity and reserve price at time t

at a price lower than the highest losing bid. Therefore, we ensure that $p = max(b_{lost}, p)$, with b_{lost} the highest losing bid.

If a bidder accepts partial fulfillment of an order, the fraction of required instances that fits in the provider's available capacity can be allocated. When multiple winning consumers are subject to such partial delivery, ties can be broken randomly.

6.8.1 Reserve Price

If profit instead of revenue is of concern, the provider needs to take its costs for delivering a VM instance into account. Let $\gamma(t)$, the *reserve price* at time t, be the lowest possible price that the provider accepts for one slot of usage of a VM instance, at time t; orders with bids below this level are ignored by the auction. In this section, we propose a method for a provider to compute $\gamma(t)$. Fig. 6.3 depicts how the order vector is shaped by $\gamma(t)$.

The reserve price for most perishable goods and services is considerably low at their expiration time. For instance, the reserve price for flight seats is theoretically negligible; as soon as boarding is closed on a particular flight, all the unsold seats on that flight are completely wasted. Thus, selling a remaining seat at a reasonable low price is often a better option compared to wasting the seat capacity without generating any revenue. However, there is a fundamental difference between cloud resources and other perishable goods and services. A significant part of the service cost in cloud data centers is related to power consumption of physical servers. The cost of power drawn by servers and associated cooling systems is comparable to the amortized capital investments for purchasing the servers themselves [88]. Thus, when considering the perishable nature of VM services, taking into account the *marginal cost* of instantiating a VM is important in this case[6].

The overall cost of the data center, $C_{overall}$, can be divided into capital and operational costs, $C_{overall} = C_{cap} + C_{opr}$. The parameter C_{cap} includes upfront investments and all one-time expenses that are depreciated over the lifetime of the data center, e.g., those related to the purchase of land, buildings, construction, buying physical servers and software, installing power delivery and cooling infrastructures etc. C_{opr} includes electricity costs, staff salaries and ISP costs. Operational costs can further be categorized as being *fixed* or *variable*, $C_{opr} = C_{opr_{fixed}} + C_{opr_{var}}$. The parameter $C_{opr_{fixed}}$ includes costs that remain identical no matter the data center is operating at full capacity or not, e.g., staff salaries. However, components of $C_{opr_{var}}$ may increase or decrease depending on data center utilization, e.g., electricity costs.

The provider is not able to avoid the incurrence of C_{cap} and $C_{opr_{fixed}}$, whereas $C_{opr_{var}}$ can be avoided to a large extent. $C_{opr_{var}}$ over any specific time period is dominated by the cost of power consumption, C_{pwr}, and can be strongly approximated by it ($C_{opr_{var}} \approx C_{pwr}$).

Cloud providers are able to measure instant power consumption in the data center. Knowing the power consumption and electricity prices, C_{pwr} can be easily calculated. We argue that the cloud provider should define the reserve price in a way that accommodating a VM with a specific bid must at least generate sufficient revenue to offset the contribution of VM to $C_{opr_{var}}$. Assuming all VMs are of the same type, $\gamma(t)$ can therefore be derived as follows:

$$\gamma(t) = C_{pwr}/VM_n(t), \tag{6.10}$$

where $VM_n(t)$ is the number of running VMs in the data center at time t, and C_{pwr} is the cost of power consumption at that time. Knowing the electricity price, φ, and total data

[6]In economics, the marginal cost is the change in total cost that arises as a result of one additional unit of production.

center power consumption, $Power_{total}$, C_{pwr} can be computed as $C_{pwr} = Power_{total} \times \varphi$. As $\gamma(t)$ is primarily affected by factors such as IT load, electricity price, data center outside air temperature and humidity [89], it should vary dynamically.

Because we resort to simulation for the experimental performance evaluation of our proposed solution, we require a model for C_{pwr}. Detailed modeling of data center power usage however is difficult because of the complexity and diversity of the infrastructure [89]. Consequently, we propose abstract model based on the concept of Power Usage Effectiveness (PUE).

6.8.2 Power Usage Efficiency Model

PUE is a measure of how efficiently a data center consumes its power. It is computed as the ratio of total data center power consumption, $Power_{total}$, to IT load power, $Power_{IT}$, i.e., power consumed by servers, storage and network equipment:

$$PUE = Power_{total} / Power_{IT}. \tag{6.11}$$

PUE measures the power overhead consumed in supporting the IT load. The overhead is caused by cooling and humidification systems (e.g., chiller), power distribution (e.g., PDU), power conditioning system (e.g., UPS), and lighting. Ideally $PUE = 1$. Inefficient data centers have a PUE of 2.0 to 3.0, while PUE scores lower than 1.14 are advertised by leading companies such as Facebook and Google [41]. PUE reported in this way is usually an average value over a specific period of time (e.g., one year), whereas instant PUE is not a constant value. The efficiency of the data center varies over time by changes in the data center conditions.

One of the most important conditions is the outside ambient temperature [97], as the energy required to remove heat generated within the data center grows with it [89]. To some degree, outside air humidity affects cooling power as well, but we do not consider it in this work in order to limit model complexity.

A second important condition that changes over time and affects PUE is the IT load. This follows from the fact that the efficiency of power conditioning system and cooling equipments increases under higher load [42]. We represent IT load by the percentage of ON physical servers in the data center (referred to as *data center utilization*). We model PUE as function of load and outside ambient temperature, i.e., $PUE = f(load, temp)$. In order to simplify the model, we assume that every server in the data center consumes its peak load power if it is ON; and none otherwise. $Power_{IT}$, is therefore computed according to (6.12).

$$Power_{IT} = N_{Srv-ON} \times Power_{Srv}, \tag{6.12}$$

where N_{Srv-ON} is the number of non-idle servers in the data center, and $Power_{srv}$ is the peak power consumption by servers. The contribution of networking equipment in (6.12) is not taken into account as it is small and its power draw does not vary significantly with data center load [89].

In this study, we assume that the provider commits to provide the actual amount of resources required by a VM, regardless of the actual resource usage pattern of the applications it executes. Moreover, we assume the cloud provider periodically packs the data center's workload into a minimum number of servers, powering off any inactive ones.

6.9 Auction Mechanisms and Benchmarks

In this section we review the different auction mechanisms that are included in our experimental evaluation.

Optimal Single Price Auction (OPT): The extractable revenue in a single-round, single-price auction is at most $\mathcal{F}(\mathbf{d})$ which can be achieved by an optimal price choice. Since we are interested in maximizing the provider's revenue, we use the Optimal Single Price Auction (OPT) described in Definition 1 as a benchmark. In the online version of OPT, the auction is executed upon every arrival of an order or termination of an instance.

Online Extended Consensus Revenue Estimate Auction (Online Ex-CORE): Details of the Ex-CORE auction can be found in Section 6.7. Our online version of Ex-CORE (outlined in Algorithm 3) records the optimal sale price computed by OPT in the previous round, and updates the sale price using the Ex-CORE algorithm. Only when the optimal

Algorithm 3 The Online Ex-CORE Auction

```
Input: d, $p_{cur}$, $p_{optprv}$    ▷ d is the list of orders, sorted in descending order of bids, $p_{cur}$ is
    current market price, $p_{optprv}$ is the optimal single price in the previous round.
Output: $p$                                                         ▷ Sale Price
 1: $p_{opt} \leftarrow opt(\mathbf{d})$
 2: if $p_{opt} = p_{optprv}$ then
 3:     return $p_{cur}$
 4: end if
 5: $r \leftarrow$ the largest $r_i$ in d
 6: $m \leftarrow \underset{\sigma_i(\mathbf{d})}{\text{argmax}} \; b_i\sigma_i(\mathbf{d})$
 7: if $m \leq r$ then
 8:     return $p_{opt}$                                            ▷ single optimal price
 9: else
10:     $\rho \leftarrow \frac{m}{m-r}$
11:     Find $c$ in $\rho\, ln(c) + \rho - c = 0$
12:     $u \leftarrow rnd(0,1)$                          ▷ chosen uniformly random on [0,1]
13:     $l \leftarrow \lfloor log_c(\mathcal{F}(\mathbf{d})) - u \rfloor$
14:     $R \leftarrow c^{(l+u)}$
15:     $j \leftarrow$ the largest $k$ such that $\frac{R}{\sigma_k(\mathbf{d})} \geq b_k$
16:     return $\frac{R}{\sigma_j(\mathbf{d})}$
17: end if
```

sale price calculated in the current round differs from the one in the previous round of the auction, a new price is computed (lines 1-4). This prevents the market to be exposed to a high number of price fluctuations due to randomness in the Ex-CORE algorithm. Note that it does not violate a possibly existing consensus established in the previous round of the Ex-CORE auction, as arriving or leaving orders have not changed the optimal price.

Lines 5 and 6 compute r, the maximum number of requested units in the order list, and m, the maximum number of units sold by OPT. As our mechanism is designed to work for mass-market scenarios it requires m to be larger than r ($m \gg r$). In the rare event when this condition would not hold, the algorithm returns the price computed by OPT.

On line 10, ρ is computed, followed by the computation of the optimal value for c, for which we use Newton-Raphson. Subsequently, c is used to generate an estimation of $\mathcal{F}(.)$ that achieves the consensus with high probability (lines 12-14). Finally, the estimated value is converted to the market clearing price through the revenue extraction mechanism.

Holding Time Aware Optimal Auction (HTA-OPT): Due to a lack of prior knowledge on the holding time of VMs, the online version of the Ex-CORE auction operates in a greedy manner, as it attempts to maximize revenue given the newly arriving order and the existing orders at a given time. In order to quantify the efficiency loss caused by this lack of information, we use HTA-OPT as a benchmark algorithm that uses prior knowledge on VM holding times. HTA-OPT takes into account the fact that an order with a long holding time and a low bid can potentially generate more revenue than a short order with a high bid.

Algorithm 4 calculates the optimal sale price using dynamic programming. The price is computed based on the maximum possible revenue that can be generated by current orders in the system and the corresponding remaining time of these orders. The main reasoning is that if the algorithm sets the price at a specific bid price, all orders with bid prices lower than that price are not available for the next time slot. Assuming bidders are charged on an hourly basis of VM usage, we express duration similarly in an hourly basis. Each partial hour is considered as a full hour (e.g., 2.5 hours is considered as 3 hours of usage).

Algorithm 4 has the following input arguments: the list of orders **d**, sorted in descending order of bids, an order index i set to the number of orders in the first call of the function, a time slot index t set to 1 for the first call, and a boolean argument $firstCall$ indicating that it is the first call to the function. Lines 5-15 initialize the revenue array rev such that each element in rev is set to the revenue that can be generated in that time slot, provided that the price is set to a corresponding bid price. Line 16 ends the recursion when the termination conditions are reached.

In lines 24-29, the algorithm chooses the most profitable path given two choices for dealing with order i at time t. This is done by recursively computing the total revenue in case the market price is kept below the order's bid at time t ($ans1$), and computing the revenue in case the decision is made to let the market price exceed the order's bid at t ($ans2$).

The most profitable decision path is stored in the dp array with the aggregated revenue for the checked paths. Finally, we find the highest possible revenue within the first column of dp, and return the corresponding price. We break ties by favoring the market price with the lowest transaction volume.

Uniform Price Auction: In the uniform price auction, the provider serves the highest bidder first, allocating the requested number of instances. This is followed by an allocation for the second highest bidder and so forth until supply is exhausted or there are no more orders. All bidders are charged the lowest winning bid.

6.10 Performance Evaluation

Our evaluation of the proposed auction framework includes three parts. In the first, we simulate Ex-CORE in a single-round, unlimited supply setting using several order distributions. In the second part, the impact of misreporting the number of required instances on the utility obtained by an individual bidder is explored. The last part evaluates the auction framework under bounded supply. Auctions then occur recurrently by arriving and finishing orders, and the marginal cost of VM production changes dynamically over time.

6.10.1 Order Generation

Due to the lack of real-world data on bidder valuations and order sizing, we need to resort to a synthetic generation of orders. In line with [39], we adopt the following four distributions for the generation of bids:

1. **uniform (l, h)**: Bid prices are drawn from a uniform distribution bounded by l and h.

2. **normal (μ, σ)**: Bid prices are drawn from a normal distribution with mean μ and standard deviation σ. Bids less than or equal to zero are discarded and a new bid is drawn from the distribution. This causes the normal distribution to be skewed as zero and negative bid values are not permitted.

Algorithm 4 Holding Time-Aware Optimal Auction

```
Input: $\mathbf{d}, i, t, firstCall$
Output: $p$                                                    ▷ Sale Price
 1: $maxDuration \leftarrow \max_i(\text{DURATION}(d_i))$      ▷ The duration function returns the time
    remaining from the holding time of the orders in unit of hours.
 2: $dp[|\mathbf{d}|][maxDuration] \leftarrow \{-1\}$
 3: $rev[|\mathbf{d}|][maxDuration] \leftarrow \{0\}$      ▷ Create $dp$ and $rev$ arrays with $|\mathbf{d}|$ (size of $\mathbf{d}$) rows
    and $maxDuration$ columns, and initialize all cells with $-1$ and 0 respectively.
 4: function HTA-OPT($\mathbf{d}, i, t, firstCall$)
 5:     if $firstCall$ then
 6:         for $j \leftarrow 1$ to $maxDuration$ do
 7:             $prvCount \leftarrow 0$
 8:             for $k \leftarrow 1$ to $|\mathbf{d}|$ do
 9:                 if $j \leq \text{DURATION}(d_k)$ then
10:                     $rev[k][j] = d_k \times (r_k + prvCount)$
11:                     $prvCount \leftarrow prvCount + r_k$
12:                 end if
13:             end for
14:         end for
15:     end if
16:     if $i = 0$ or $t > \text{DURATION}(d_i)$ then
17:         return 0;
18:     end if
19:     if $dp[i][t] = -1$ then
20:         $ans1 \leftarrow 0, ans2 \leftarrow 0$
21:         if $t \leq \text{DURATION}(d_i)$ then
22:             $ans1 \leftarrow \text{HTA-OPT}(\mathbf{d}, i, t+1, false) + rev[i][t]$
23:         end if
24:         if $i \geq 1$ then
25:             $ans2 \leftarrow \text{HTA-OPT}(\mathbf{d}, i-1, t, false)$
26:         end if
27:         if $ans1 > ans2$ then $dp[i][t] \leftarrow ans1$
28:         else $dp[i][t] \leftarrow ans2$
29:         end if
30:     end if
31:     if $firstCall$ then
32:         $k \leftarrow \text{argmax}_i(dp[i][0])$      ▷ In case of ties, pick lowest $i$, i.e., higher price and
    selling less instances
33:         return $b_k$
34:     end if
35:     return $dp[i][t]$
36: end function
```

3. **Zipf (h, θ)**: Bid prices are drawn from a Zipf distribution with parameters h as the highest bid price and parameter θ. This distribution is a generalization of the Pareto principle that 80% of the total bid value originates from 20% of the bidders.

4. **bipolar (l, h)**: Bid prices are generated by randomly choosing either l or h with equal probability.

For requested number of instances in each order, we consider three types of distributions :

1. **constant(ζ)**: The number of instances for all orders equal $\zeta \leq r$ where r is the supremum on the number of requested units.

2. **uniform (l, h)**: The number of instances for an order is drawn from a uniform distribution between $l = 1$ and $h = r$.

3. **normal (μ, σ)**: The number of instances for an order is drawn from a normal distribution as a discrete value with mean μ and standard deviation σ. Values smaller than 1 or larger than r are discarded.

6.10.2 Single Round Evaluation

We investigate the generated revenue for different combinations of distributions for the bid price and the number of requested units per order. The number of orders varies between 10 and 100000 and the ratio of generated revenue by Ex-CORE (R) to $\mathcal{F}$ is reported ($R/\mathcal{F}$). Each experiment is carried out 30 times and the mean value of $R/\mathcal{F}$ is reported. Fig. 6.4 shows the simulation results when $r = 50$. As the number of orders increases, $R/\mathcal{F}$ approaches 1 regardless of the distribution used for order generation as we expected. Although there is a small difference between the revenue obtained by Ex-CORE for different distributions as shown in Fig. 6.4, the distribution of orders does not have a significant effect on the generated revenue, especially when the number of orders in the market is large. Fig. 6.5 shows separate box plots of $R/\mathcal{F}$ for different order distributions when the number of orders equals 100. Statistical analysis certifies that the performance does not change significantly under different order distributions. By design, Ex-CORE

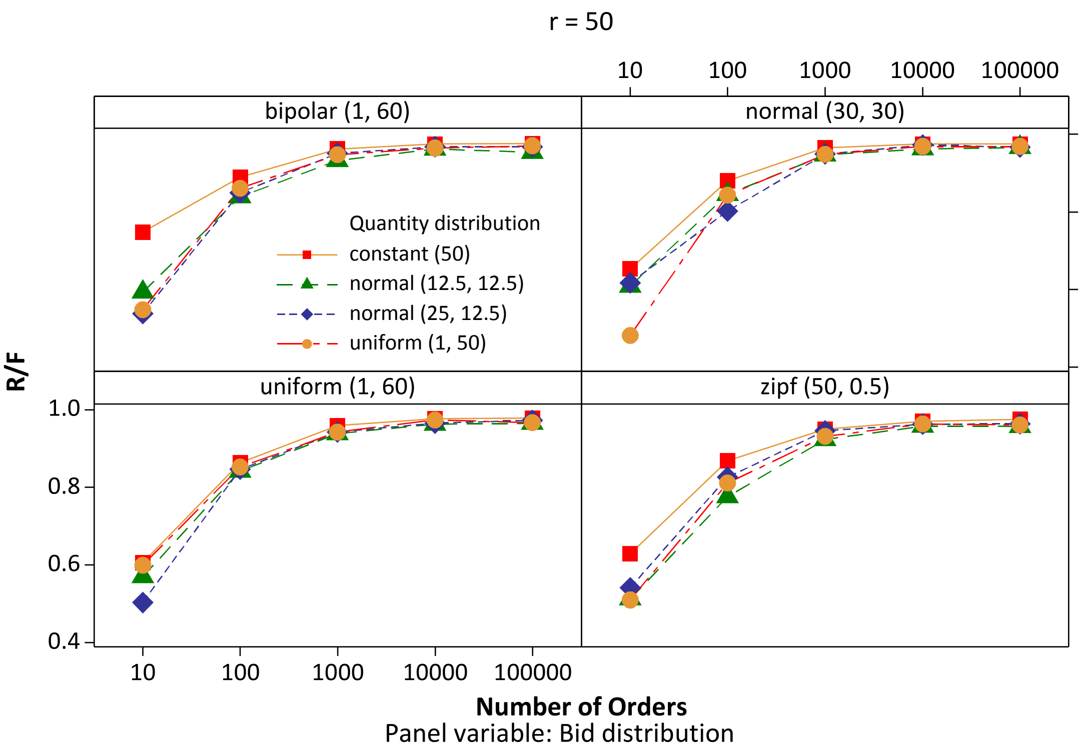


Figure 6.4: Ratio of gained revenue by the Ex-CORE auction to optimal auction under different distribution of orders.

does not require a priori knowledge about the order distribution. Therefore, the provider does not need to rely on frequent investigation and monitoring of changes in the market conditions in order to maximize its revenue.

A sensitivity analysis with respect to r showed that its value does significantly impact the results, as long as r is sufficiently small compared to the total demand volume, and the total supply volume in the market is sufficiently large. This can be easily justified by Lemma 3. For brevity, we omit a discussion on these experiments.

6.10.3 Evaluation of Misreporting Quantity

As the Ex-CORE mechanism is not two-dimensionally truthful, we investigate the potential for a bidder to gain utility by misreporting the number of required units in her order. First, we generate list of orders with the same settings used for experiment 1 and assume each generated order to be truthful. The utility obtained by every bidder is subsequently

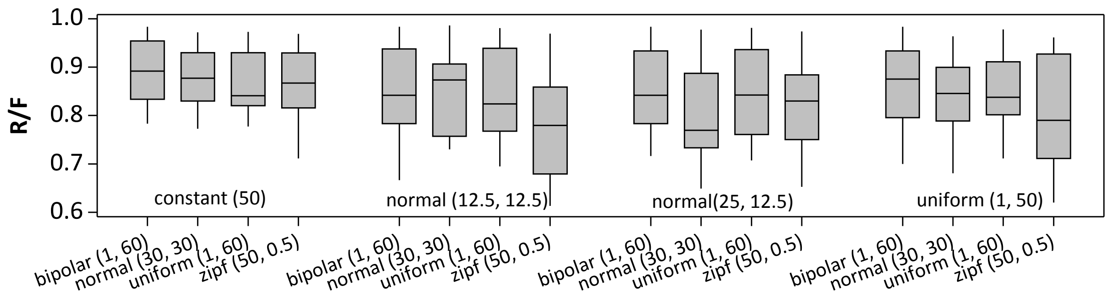


Figure 6.5: Ratio of gained revenue by the Ex-CORE auction to optimal auction under different distribution of orders when number of orders is 100.

calculated according to (6.1).

Assuming there is no collusion among bidders, we increase the requested number of instances (r_i) for an individual bidder i up to the maximum number of instances that can be requested (r) by the step size of one, while keeping the orders of the other bidders unchanged. For every stepwise increase, we calculate the bidder's utility and compare it with the utility attained under a truthful report. We then compute the probability of a bidder increasing its utility through a misreport of quantity, by dividing the number of cases in which utility increased to the total number of steps.

The experiment is repeated for every bidder with the same random number seed for each step and is subsequently carried out 30 times with different seeds. Fig. 6.6 shows the mean and box plot of the mean probability of increase in the utility by misreporting the quantity after 30 runs under different order distributions. The constant distribution of the requested units is removed from the set of quantity distributions, as there is no opportunity to change r_i. As one can observe in the figure, the probability of gaining utility through misreports converges to zero as the market size grows under all order distributions. Moreover, there is no predictable pattern for the user to increase the utility value due to the implicit random component of Ex-CORE and the lack of knowledge about other bidders. Fig. 6.7 shows the maximum increase of the utility value among all bidders, achieved through misreporting quantity. As can be seen in the figure, the maximum possible gain in the utility value for all bidders through misreporting number of instances converges to zero as the market size grows.

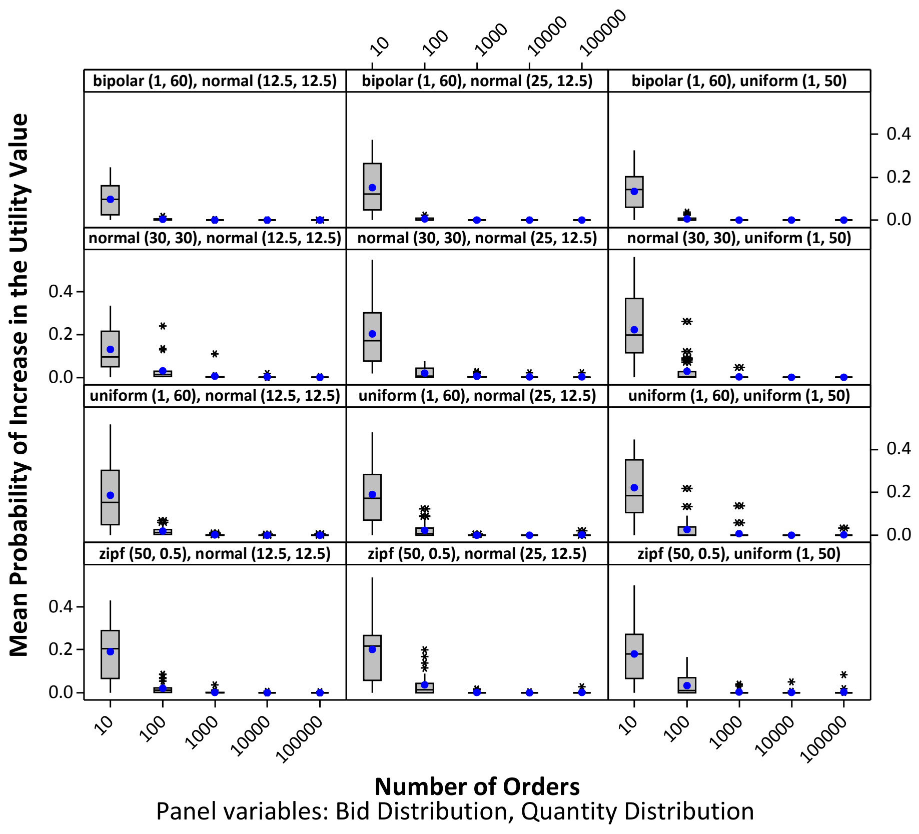


Figure 6.6: Mean probability of increase in the utility value for bidders by Ex-CORE under different distribution of orders when $r = 50$. A blue circle denotes the mean value.

6.10.4 Online Auction Framework Evaluation

We evaluate the profit of the online Ex-CORE auction through simulation. We consider the case where capacity (C) is bounded fixed throughout the simulation at $C = 8 \times 10^4$. In real-world scenarios, a provider may offer several pricing plans (e.g., on-demand, reserved pricing plans) or different types of VMs (e.g., small, medium, large). Under such circumstances, the capacity allocated for a specific VM type in the auction market can be dynamically adjusted in order to maximize profit [127,139]. We consider the auction operates for just one type of VM. Similar auctions can be run separately for the provider's VM types.

We simulate the market for 24 hours. Customers submit their orders independently following a Poisson process with λ set at the total number of requests in the whole simulation divided by 24. As the distribution of the bid prices and the quantity of requested

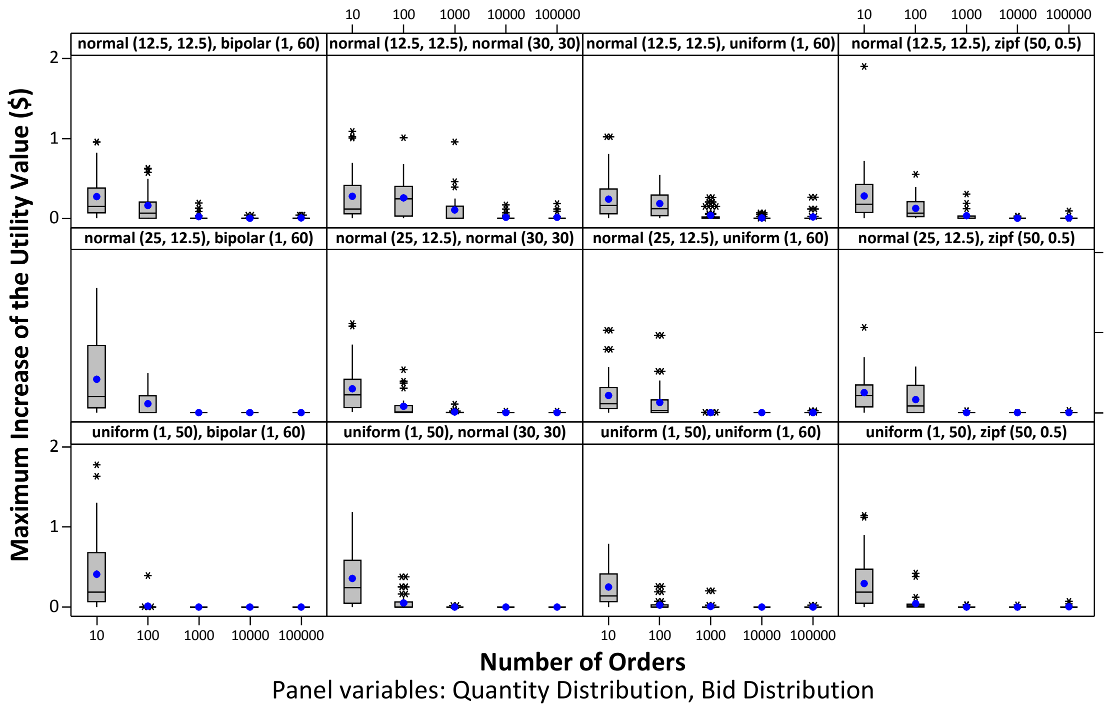


Figure 6.7: Maximum increase of the utility value among all bidders, achieved through misreporting quantity in Ex-CORE auction under different distribution of orders when $r = 50$. Mean value is denoted by blue disc.

units do not significantly impact the revenue results, we use uniform distributions for both.

Bid prices in dollars are drawn from a uniform distribution on $[0, 0.06]$. The maximum bid price is derived from the Amazon EC2 price of on-demand small instances in the US east region[7]. Considering the lower QoS for VMs in the spot market and the truthfulness properties of the mechanism, bidding higher than the on-demand price seems unreasonable. However, in real-world scenarios there might be orders with a bid higher than the on-demand price, as observed on the EC2 spot market. This is not of concern in our model.

The requested number of instances per order for each bidder is modeled by i.i.d. random variables uniformly distributed on $[1, 50]$. Amazon EC2 similarly imposes a limit of 100 VM instances per region that can be acquired by a customer in the spot market[8].

[7]http://aws.amazon.com/ec2/pricing/
[8]http://aws.amazon.com/ec2/spot-instances/

Following Mills et al. [74], the holding time of the VM instances by users is modeled by i.i.d Pareto distributed random variables, with shape parameter 1 and location parameter 1. Each generated random value represents the time in hours that VM instances remain in the system. If the order's bid price is lower than the current market price, the order remains in the queue for a maximum time period of half an hour. The order is considered in every auction round during this period. If the order is not serviced at the end of this period, it is labeled as rejected. The VMs that are instantiated following the acceptance of an order can be terminated at any time if the market price exceeds the order's bid. Upon termination, these VMs are not charged for their last partial hour. VMs that are terminated by their owner are charged for a discrete number of hours, with a partial hour of usage accounted for as a full hour.

To model the marginal cost of running VMs, we assume that the data center is populated with BL460c G6 blade servers that host a quad-core Intel Xeon E5504 2.0 GHz processor. The peak power usage per blade server is rated at 400 W. Using the Amazon EC2 small instances type, each server is able to host up to 8 VMs. Two sets of electricity prices are considered for the data center, one for "on-peak" hours from 7am to 9pm and another for "off-peak" hours from 9pm to 7am. Following work by [53], we adopt a peak price of 0.108$/KWh and an off-peak price reduction of 50%. We compute PUE based on data center load and outside air temperature. Taking models by Goiri et al. [36] and Rasmussen [97] into account, Fig. 6.8 illustrates our modeling of the PUE as a function of outside temperature. Drastic jumps in PUE occur due to chiller activation when outside temperature crosses the $20°C$ mark [36]. We consider a relatively warm day with minimum temperature of $14°C$ and maximum temperature of $33°C$. We estimate hourly temperatures throughout the day based on a method by Gaylon et al. [18].

The green dashed line in Fig. 6.10 depicts the reserve price generated based on our model in a sample simulation run. Our investigation on the historical price data of spot instances for the past 90 days prior to 14th of November 2013 shows that the spot market price never goes below \$0.007 for the small instances in the US-east region. The value complies with our computed reserve price for that instance type when the physical server characteristics, as well as the electricity prices and outside air temperature parameters are

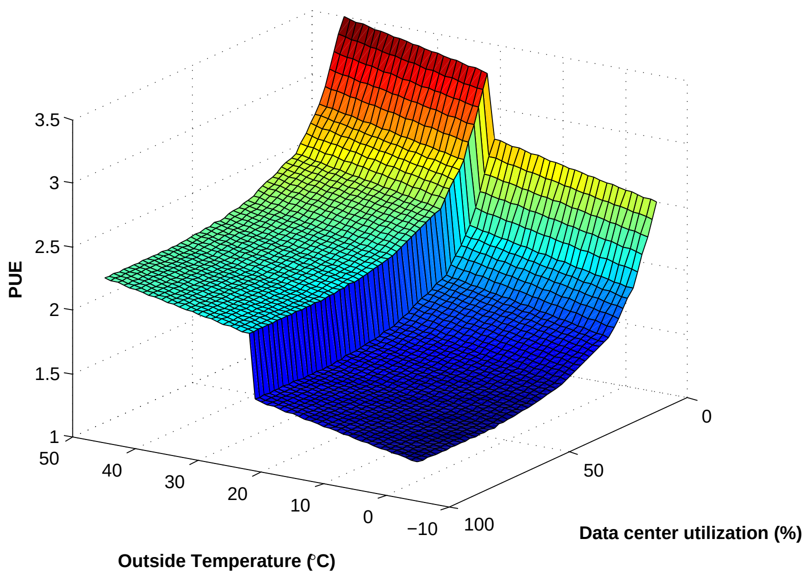


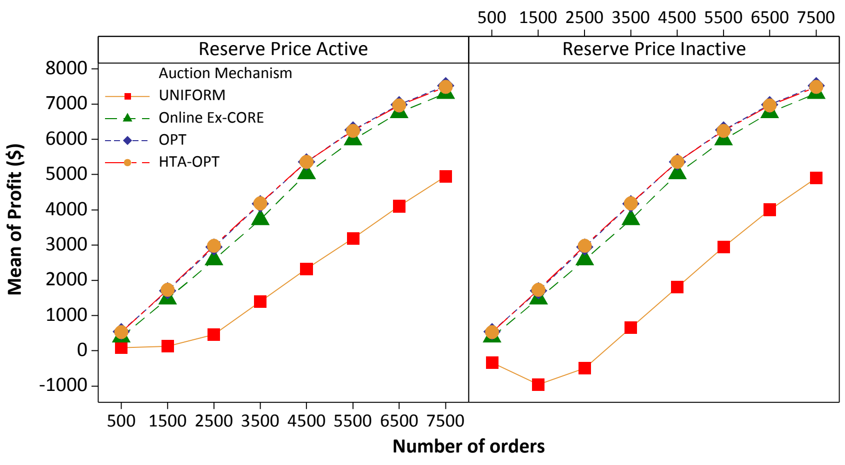
Figure 6.8: *PUE* as related to load and outside temperature.

based on realistic data for an Amazon data center in the US-East region. The same holds true for the modeled and observed minimum spot price for the other instance types in the `m1` instance class, as they are based on hardware with similar power draw characteristics.

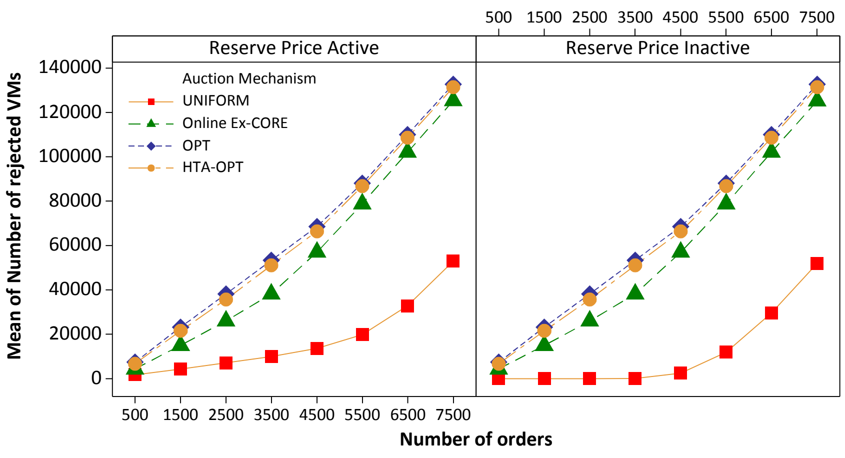
Experimental Results

We evaluate the online Ex-CORE auction by comparing profit and number of rejected VM requests to the other auction mechanisms outlined in Section 6.9. The computed profit is the total generated revenue minus the cost of electricity. The capital cost and all other fixed cost are not considered, as they are identical for all mechanisms. Each experiment is carried out 30 times and the mean value is reported. The results are illustrated in Fig. 6.9a and Fig. 6.9b, where the number of orders in a 24-hour simulation is increased from 500 to 7500, and scenarios with or without the adoption of a reserve price are shown.

Fig. 6.9a shows that gained profit by all mechanisms increases with the number of orders. The OPT, HTA-OPT and online Ex-CORE auctions generate comparable profits, while there is a big gap between uniform price auction and the other mechanisms. When supply is higher than demand and there is no competition among bidders, all orders are accepted by the uniform price auction; and consequently the uniform price auction



(a)



(b)

Figure 6.9: (a) Average profit gained and (b) number of rejected VM instances with different auction mechanisms.

performs poorly under such circumstances. This supports the idea that the traditional auction mechanisms such as the Vickrey Auction [122] or Uniform price auction are not suitable for the cloud spot market in which supply is often higher than demand.

The benefit of using the online Ex-CORE auction is that, in spite of a small difference in generated profit compared to OPT and HTA-OPT (6% lower on average), it accommodates a considerably higher number of VMs (17% and 14% less rejections on average respectively). This reduces the impact of the bidder drop problem, introduced by Lee and Szymanski [54] that can be caused by frequent rejection of customers with low valuations.

As illustrated in Fig. 6.9a and Fig. 6.9b, the reserve price only affects the outcome of the uniform price auction. Considering the range and distribution of bid values used in the simulation, the market price generated by online Ex-CORE, OPT and HTA-OPT is always higher than the reserve price. To exemplify further, Fig. 6.10 provides the reserve price and the market price generated by online Ex-CORE in a sample simulation run. This, however, does not mean that the reserve price is of no importance and can be ignored in real-world scenarios. In order to show the impact of the reserve price, different highest submitted bid prices are used to decrease the average market price as shown in Fig. 6.11. As can be seen in the figure, when the highest submitted bid price is low and therefore the market price is lower on average, the absence of reserve price can lead to loss or lower profit due to execution of VMs at a price below their variable cost.

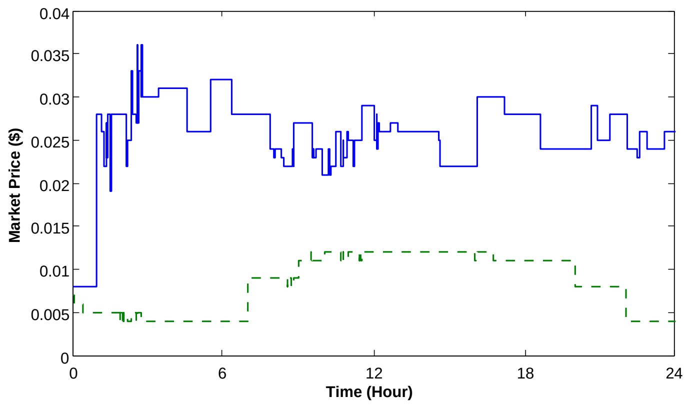


Figure 6.10: Reserve price (green dashed line) and spot market price generated by online Ex-CORE (blue solid line) in a sample simulation run when the number of orders is 1500.

As we are interested in the importance of a priori knowledge on the holding time of VMs by customers, the profit and the number of rejected VMs by OPT and HTA-OPT are investigated further. The results of a paired T-Test comparing the profit performance of OPT with HTA-OPT when the number of orders is 4500 and the holding time of VMs is distributed i.i.d. based on a Pareto distribution with both shape and location parameters equal to one are shown in Table 6.1. Given a null hypothesis of no statistically significant difference in mean profit by OPT and HTA-OPT, the p-value is relatively high (p-value = 0.846), suggesting that there is no strong evidence that the null hypothesis is false, i.e. there is no credible evidence that OPT and HTA-OPT on average generate different

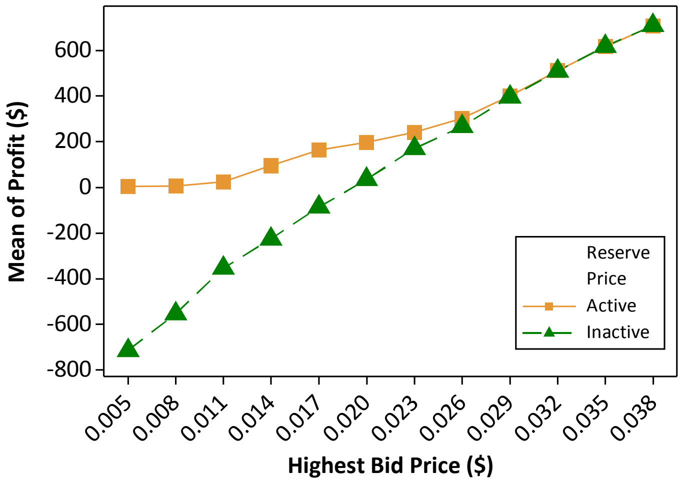


Figure 6.11: Average profit gained by Ex-CORE when the number of orders is 1500.

Table 6.1: Paired T-Test with 95% Confidence Interval (CI) for comparison of difference in mean of profit and number of rejected VMs generated by OPT and HTA-OPT (OPT − HTA-OPT) when the number of orders is 4500.

	OPT	**HTA-OPT**	**Difference (95% CI)**	**P-value**
Profit	5358.7	5361.6	-2.9 (-33.6, 27.7)	=0.846
Rejected	68575	66423	2152 (1192, 3111)	<0.001

profit. However, there is a statistically significant difference in the mean number of rejected VMs. HTA-OPT rejected 2152 less VMs on average as it results in outcomes with a lower market price. Considering the reported 95% Confidence Interval (CI), we can state that knowing the holding time of VMs by itself does not change the amount of profit a provider generates as it is not aware of upcoming orders' bid prices.

6.11 Summary and Conclusion

With the rapid adoption of cloud computing environments, balancing supply and demand for cloud resources through dynamic forms of pricing is quickly gaining importance. In this chapter, we presented an envy-free auction that is truthful with high probability and that generates a near optimal profit for the cloud provider. The auction operates under conditions similar to the EC2 spot market. The truthfulness of the mechanism frees bidders from understanding its intricacies, thereby lowering the complexity of participation and the options for strategic behavior. At the same time, the mechanism aims to achieve a maximal profit for the provider, and achieves envy-freeness through the use of a uniform price. The mechanism is a generalization and extension of the consensus revenue estimate (CORE) auction that enables its application in the cloud computing setting, which requires an online recurrent auction with multi-unit requests. In order to incorporate marginal costs of production in the resource trading process, we pair the auctioning scheme with a method that calculates dynamic reserve prices based on a cost model that incorporates data center PUE, load, and electricity cost.

An important benefit of the proposed auction design is that it achieves near optimality w.r.t. maximizing revenue without requiring prior knowledge on the bid distributions. Our evaluation demonstrates its performance in this regard under a variety of order distributions. The proposed mechanism is shown to significantly outperform the uniform price auction and to closely approximate the profit outcome of the revenue maximizing, but non-truthful, optimal single price auction in an online setting (within 6% in our experiments), while improving on the number of rejected VMs (up to 17% in our experiments). Finally, our results show that the generated revenue does not differ significantly from the revenue attained by a mechanism based on dynamic programming that relies on prior knowledge regarding the holding time of VMs.

Chapter 7

Spot Instance Pricing as a Service

The previous chapter introduced an auction mechanism for cloud spot markets that efficiently prices cloud resources in line with a provider's profit maximization goal. This chapter presents the implementation of the proposed mechanism by identifying a framework called Spot instance pricing as a Service (SipaaS)[a]*. SipaaS is an open source project offering a set of web services to price and sell VM instances in a spot market. Cloud providers, who aim at utilizing SipaaS, should install add-ons in their existing platform to make use of the framework. As an instance, we provide an extension to the* Horizon *–the OpenStack dashboard project– to employ SipaaS web services and to add a spot market environment to OpenStack. The design and implementation of the framework followed by its evaluation and validation are described in detail in this chapter.*

[a]SipaaS in Persian language means thank.

7.1 Introduction

RECENTLY, Infrastructure-as-a-Service (IaaS) cloud providers have started offering unused computational resources in the form of dynamically priced virtual machines (VM instances). The fact that demand for computational resources is non-uniform over time motivates the use of dynamic forms of pricing in order to optimize revenue. Hence, design and implementation of dynamic pricing mechanisms have received considerable attention in the literature [9, 127, 134].

Among the pioneers who sell spare capacity of data centers using an auction-like dynamic pricing mechanism is Amazon Web Services (AWS)[1]. In Amazon terminology, VM instances trading in this form of pricing is known as *spot instances* and the market in which spot instances are traded is called *spot market*.

[1]Amazon Web Services (AWS), `http://aws.amazon.com`.

Spot market, since introduced by AWS, has been considered as one of the first steps towards a full-fledged market economy for computational resources [139]. In the spot market, customers communicate their bids for an instance-hour to AWS in order to acquire required number of instances. Subsequently, AWS reports a market-wide *spot price* at which VM instance use is charged, while terminating any instances that are executing on a bid price that is lower than the market price (*out-of-bid* situation). Prices vary independently for each instance type and available data center (or *availability zone* in Amazon terminology).

AWS has revealed no detailed information regarding their pricing mechanism and the computation of the spot price. At present, the design of dynamic forms of pricing for cloud computing resources is an open research challenge, and of great interest to both cloud providers and researchers. An auction mechanism is truthful, if for each bidder and irrespective of any choice of bid by all other bidders, the dominant strategy for the bidder is to report his/her true information. We presented a pricing mechanism called *Online Extended Consensus Revenue Estimate (online Ex-CORE)* auction that is *truthful* with high probability and generates a near optimal profit for the cloud provider in Chapter 6.

In this chapter, we describe an open source framework called *Spot instance pricing as a Service (SipaaS)*. SipaaS offers a set of web services that can be used by IaaS cloud providers to run a spot market resembling the AWS spot market. It provides services to price VM instances using the internal pricing module that works based on the online Ex-CORE auction mechanism. The extensible architecture of the SipaaS framework allows for implementation of any new pricing mechanism without the necessity to modify the design of the web services.

Cloud providers, who aim at utilizing SipaaS, require to extend their platform to make use of the framework. In this chapter, we provide an extension to the OpenStack project[2], as an example, to employ SipaaS web services. Accordingly, we extended the *Horizon – the OpenStack dashboard project* – to run a spot market environment using web services provided by SipaaS.

To validate and evaluate the system consisting of the SipaaS framework combined

[2]OpenStack: An open source software for building private and public clouds, `http://www.openstack.org/`.

with the extension to OpenStack, we conducted an experimental study with a group of ten participants utilizing the provided spot market. Results show that the system performs perfectly in a practical test environment and experimentally confirm the theoretically proven truthfulness feature of the Ex-CORE auction.

The remainder of this chapter is organized as follows: Section 7.2 discusses the system design and implementation where we describe SipaaS, extensions to Horizon, and Ex-CORE pricing mechanism in subsections 7.2.1, 7.2.2, and 7.2.3, respectively. Evaluation and validation of the system is conducted in Section 7.3. Conclusions are presented in Section 7.4.

7.2 System Design and Implementation

The aim of SipaaS is to provide an extensible framework for dynamic pricing of VM instances in a spot market based on a set of RESTful services. By extensibility, we mean the ability to implement new pricing mechanisms and apply them in the framework without the necessity to modify the design of the web services. Different implementations of pricing mechanisms can be plugged into the framework by replacing the pricing module, which will be discussed in the later part of this chapter. The implementation of the SipaaS framework encompasses the proposed auction mechanism in Chapter 6 that can be easily replaced by any arbitrary dynamic pricing mechanism.

SipaaS provides services for adding, removing, or updating bidders' orders (bid price and quantity) for various types of VMs for each provider (or data center) and dynamically computing prices for each type. The SipaaS framework considers each type of VM for each cloud provider as a distinct spot market and computes prices in each market based on the submitted orders for the corresponding type. The framework is agnostic to the cloud platform and resource management system used by the cloud provider; therefore, cloud providers are supposed to implement their own extension to make use of services provided by SipaaS.

As shown in Figure 7.1, the system is composed of two main parts:

1) *SipaaS Framework*: A set of RESTful services written in Java and deployed on a

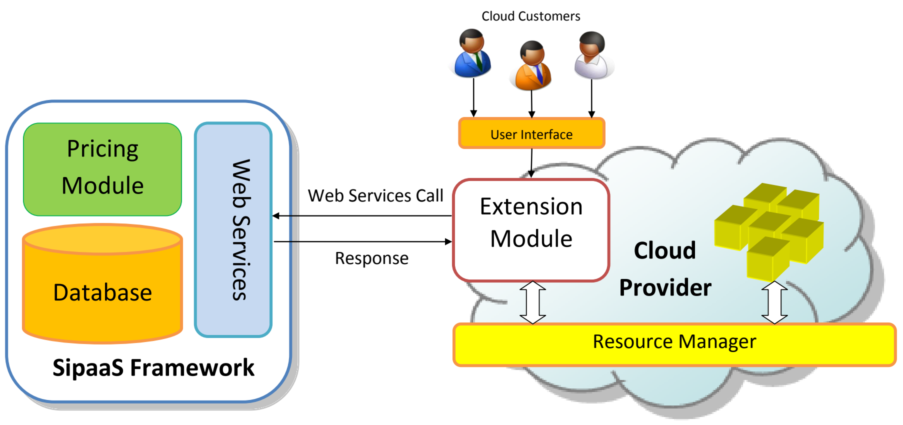


Figure 7.1: System Model.

host to provide web services required for running the spot market. Detailed information about the web services SipaaS offers are presented in Section 7.2.1.

2) *Cloud Provider's Platform Extensions*: An add-on software that must be installed as a module on the cloud provider's platform to make use of web services provided by SipaaS. Detailed information on such an extension to OpenStack is presented in Section 7.2.2.

As shown in Figure 7.1, add-on software on the provider's platform calls web services on SipaaS framework using the REST API (i.e., HTTP Requests) and receives responses in JSON [24] format in case of successful calls or error messages in case of errors.

7.2.1 SipaaS Framework

SipaaS stands for Spot instance pricing as a Service. As its name implies, the main goal of SipaaS is to provide pricing for spot instances in the form of services. Thus, it has been designed based on a set of web services, by invoking them, the cloud provider is able to price the spot instances. SipaaS web services are implemented based on Spring MVC [48]. As shown in Figure 7.2, SipaaS contains three main components:

1. *Pricing Module*: This component is the heart of the system and embodies the technique used for pricing spot instances. The pricing module computes the market-wide single price based on the submitted orders by customers. The pricing tech-

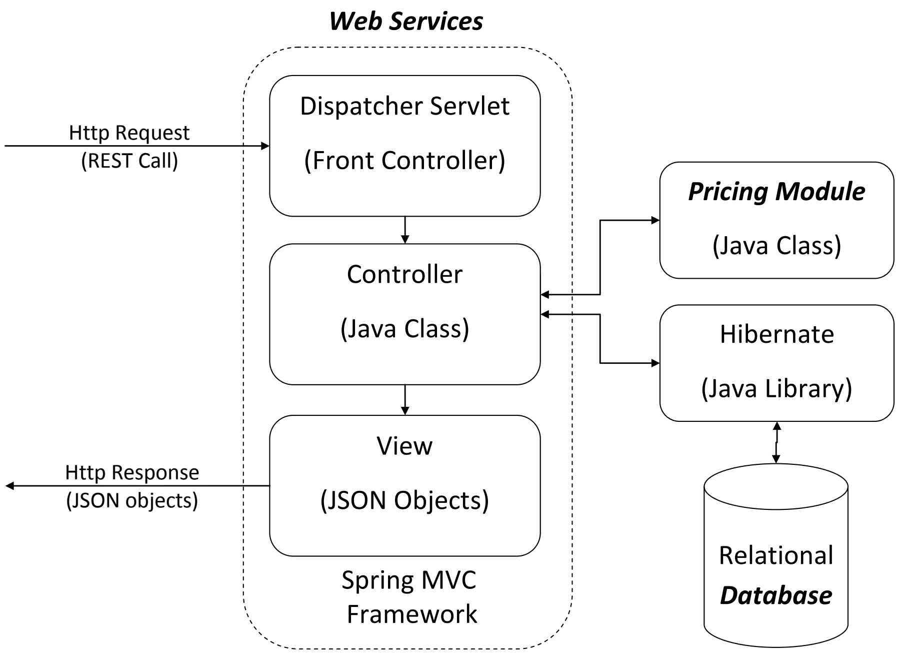


Figure 7.2: SipaaS Framework Components.

nique used in SipaaS is designed and implemented according to the proposed auction mechanism in Chapter 6. The pricing module receives a list of orders with the *reserve price* and the *available capacity* in number of VMs, which have been set by the provider and computes the spot market price accordingly. Details of auction mechanism employed in SipaaS are given in Section 7.2.3.

2. *Database*: A relational database is utilized by SipaaS to store information related to each spot market. MySQL is chosen as a *Database Management System* (DBMS), which can be replaced by any other type of DBMS according to the requirements. The database server can be deployed either on the same host where SipaaS is installed or on a dedicated host. Figure 7.3 depicts the *Enhanced Entity Relationship Diagram* (EERD) of the SipaaS' database. As it is shown in the figure, the database contains 8 main tables:

   (a) *provider*: The *provider* table stores information about providers (or data centers) which register themselves in SipaaS. Each provider has a unique *id*, *accesskey*, and *secretkey*. The provider might have any arbitrary *name* as well. Providers require their *accesskey* and *secretkey* to invoke web services provided by SipaaS.

(b) *vmtype*: Providers might have different types of VMs for each of which a distinct spot market executes. The *vmtype* table stores information about these types for every provider. The information contains: a unique id (*id*), provider's id (*provider*) and the type name (*type*).

(c) *order*: The *order* table stores information about orders by customers for each VM type and each provider. The information contains: a unique id (*id*), provider's id (*provider*), VM type id (*type*), the requested number of VMs (*qty*), bid price (*bid*) and a unique reference id (*bidref*) which must be generated by the cloud provider and is used for any future reference to this order.

(d) *price*: The *price* table stores information about spot market price generated by the pricing module. Each price contains *id*, *provider*, *vmtype*, *time*, and *price*. The *time* attribute records the timestamp for a given date and time of day the price is computed.

(e) *available*: The *available* table stores data on available capacity of the provider for each spot market. The table contains: *id*, *provider*, *vmtype*, *amount*, and *time*. The *amount* and *time* attributes are used to store the total number of available VMs for the corresponding VM type and timestamp the available capacity is set respectively. If the available capacity is not set by the provider, SipaaS assumes infinite availability.

It is worth noting that as demand for each type of VMs can fluctuate over time, providers are supposed to dynamically adjust the capacity allocated to each spot market to match the demand in order to maximize total revenue. In the current implementation of the SipaaS framework, the provider is responsible for adjusting the spot market capacity and continuously updating the availability if demand surpasses supply. One possible future extension can be adding components to handle the dynamic capacity control for each spot market. Work similar to that of Zhang et al. [139] would be useful in this regard.

(f) *reserveprice*: The *reserveprice* table, similar to *available* table, stores data on the reserve price for each spot market. Reserve price is the lowest bid price that

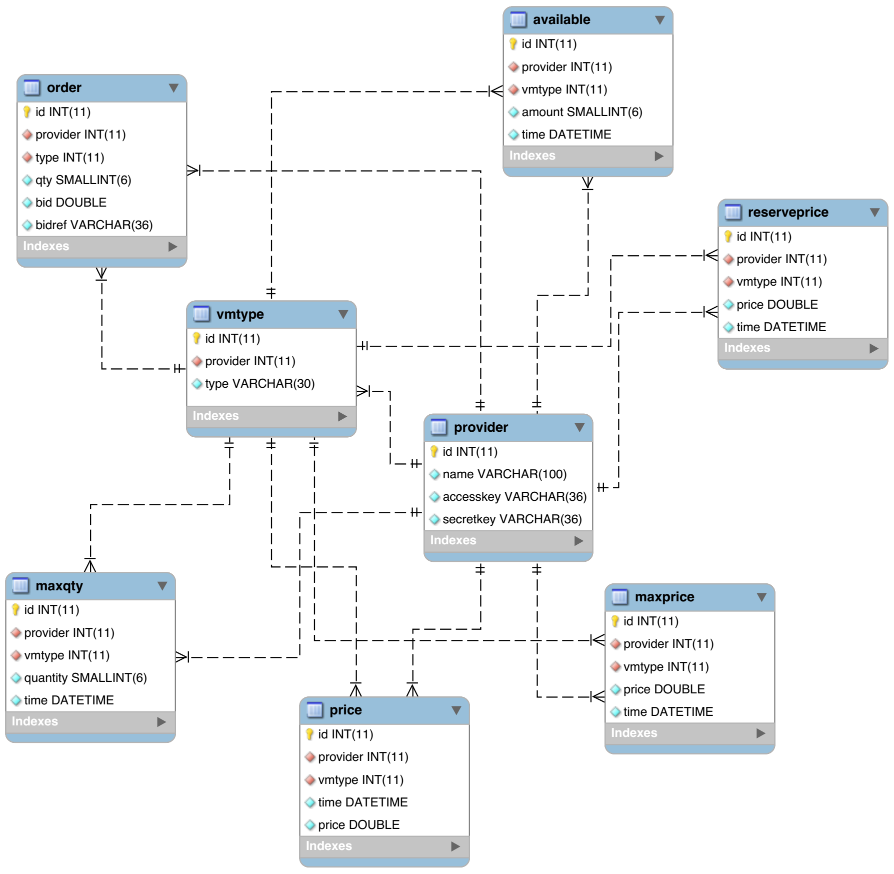


Figure 7.3: EERD of the Database.

the provider accepts for a time slot of VM instance usage (e.g., instance-hour). This table contains: *id*, *provider*, *vmtype*, *price*, and *time*. If the reserve price is not set by the provider, the minimum value of zero is assumed.

(g) *maxprice*: The *maxprice* table stores the maximum bid price acceptable by the pricing module. This table contains: *id*, *provider*, *vmtype*, *price*, and *time*. No maximum bid price suggests no limit on the bid price.

(h) *maxqty*: The *maxqty* table stores maximum number of VMs that can be requested by a customer (i.e., an order). This table contains: *id*, *provider*, *vmtype*, *quantity*, and *time*.

3. *Web Services*: SipaaS provides a set of web services that facilitate the execution of spot markets by cloud providers. Table 7.1 shows the list of main web services

Table 7.1: SipaaS Framework Web Services.

Web Service	Input Parameter(s)	Output
`register`	`name`	`credentials`
`unregister`	`accesskey,secretkey`	`status`
`regvmtype`	`accesskey,secretkey,type`	`status`
`unregvmtype`	`accesskey,secretkey,type`	`status`
`setavailables`	`accesskey,secretkey,vmtype,quantity`	`price`
`setmaxqty`	`accesskey,secretkey,vmtype,quantity`	`status`
`setreserveprice`	`accesskey,secretkey,vmtype,value`	`price`
`setmaxprice`	`accesskey,secretkey,vmtype,value`	`status`
`addorder`	`accesskey,secretkey,vmtype,quantity,bid,ref`	`price`
`updateorder`	`accesskey,secretkey,quantity,ref`	`price`
`removeorder`	`accesskey,secretkey,ref`	`price`
`pricehistory`	`accesskey,secretkey,vmtype,fromdate,todate`	`price(s)`

available in SipaaS. All services are RESTful web services designed to produce responses in the JSON format. SipaaS utilizes RESTful web services as they are easy to implement and minimal middleware is necessary, that is, only HTTP support is required. JSON is also a highly portable data transfer format that can be easily recognized by client applications. The cloud provider aiming at utilizing SipaaS framework services requires a clear understanding of the context as well as the content that must be passed through each web service invocation. The following parameters are mostly common among different web services:

(a) *accesskey*: is an alphanumeric text string that is uniquely assigned to the provider and identifies its owner. This parameter is used to differentiate cloud providers from each other.

(b) *secretkey*: plays the role of a password for the provider. A *secretkey* with *accesskey* form a secure information set that confirms the provider's identity.

(c) *vmtype*: determines the type of VM or equally a specific spot market. The *vmtype* is a string containing the VM type name and is used to relate each request to the corresponding spot market.

Below we describe the main web services provided by the framework:

(a) *register*: This service allows the provider to register itself with framework.

It receives one string parameter called *name* as an input representing the provider's name. In response to successful registration, the web service generates as output a pair of *accesskey* and *secrectkey* in JSON format.

(b) *unregister*: The provider is able to unregister from SipaaS by invoking this web service.

(c) *regvmtype*: Using this web service, the cloud provider is able to register different types of VMs in the system. In addition to *accesskey* and *secretkey*, another input parameter called *type* must be provided. The *type* parameter is a string value representing the VM type name. As stated earlier, each VM type together with a provider stands for a distinct spot market.

(d) *unregvmtype*: As its name implies, it removes a VM type.

(e) *setavaialables*: This web service receives *vmtype* and *quantity* as inputs to specify the maximum available capacity in number of VMs for the specific spot market. The *setavaialables* web service can be called any time and multiple times throughout the spot market lifetime. The output of this web service is a spot market *price* computed according to the updated capacity.

(f) *setmaxqty*: It performs similarly to *setavailables*, whereas it specifies the maximum number of VMs user can request in one order.

(g) *setreserveprice*: This web service performs similar to *setavaialables* and specifies the reserve price. It is important to mention that invoking *setreserveprice* and *setavaialables* might not result in a new spot market price, in that case, the JSON response includes the same spot price as the latest one.

(h) *setmaxprice*: It specifies the maximum bid price allowed in an order. By using *setmaxprice* and *setreserveprice*, a provider is able to limit the range of bid prices submitted by spot market users.

(i) *addorder*: The *addorder* web service allows providers to insert a new order (*bid price* plus *quantity*) to the system. The *ref* parameter is a unique value provided for each order and is used for future references to this order. The output of the service is the spot market *price*.

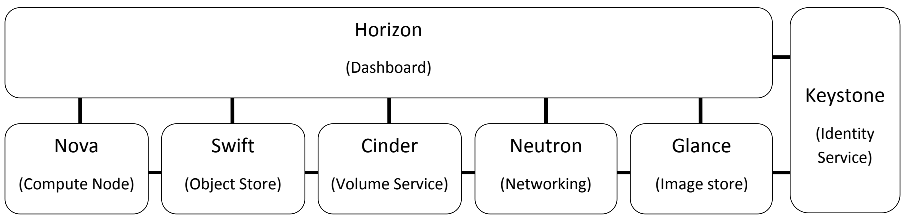


Figure 7.4: Openstack components.

(j) *updateorder*: The *updateorder* web service allows providers to update a previously submitted order to the system. This web service is called when a customer terminates part of requested running instances under the specific order.

(k) *removeorder*: The *removeorder* web service allows provider to remove a previously submitted order. This web service is called whenever all VM instances of the accepted order are terminated.

(l) *getpricehistory*: This web service receives the input parameters *fromdate* and *todate* and in response provides the pricing history of a certain spot market.

7.2.2 Extensions for Horizon - The OpenStack Dashboard

As stated earlier, cloud providers which are interested in utilizing the SipaaS framework need to setup their own customized extension software capable of interacting with SipaaS web services. In this section, we discuss about such an extension designed and implemented for the *OpenStack* platform.

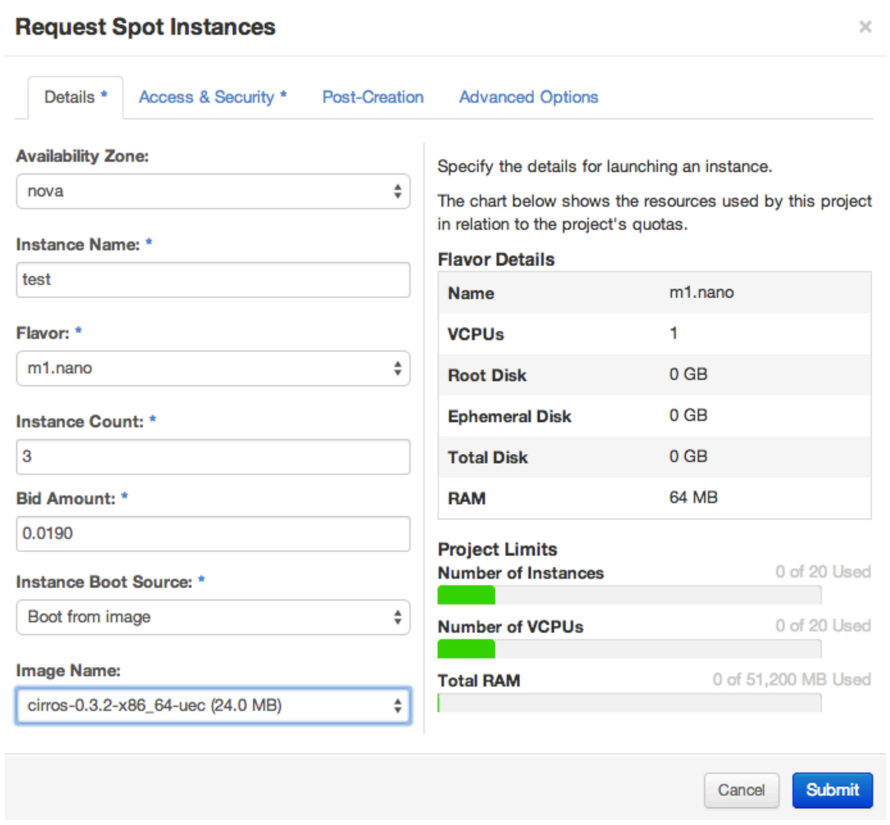
OpenStack is an open-source cloud management platform, developed to control pools of compute, storage, and networking resources in a data center. OpenStack has been designed as a series of loosely coupled components that are easy to integrate with a variety of solutions and hardware platforms. One of these components is *Horizon*, which provides users and administrators with management capabilities via a web interface. As schematically shown in Figure 7.4, Horizon provides a dashboard interface for accessing OpenStack services provided through its main components. In the following, we briefly describe the main components of OpenStack platform:

- *OpenStack Dashboard (Horizon)*: It provides a web based user interface to other services such as Nova, Swift, and Keystone. Management actions enabled by this component include VM image management, VM instance life cycle management, and storage management;

- *OpenStack Compute (Nova)*: It manages the VM instance life cycle from scheduling and resource provisioning to live migration;
- *OpenStack Storage (Swift)*: Swift is a scalable redundant storage system responsible for enabling data replication and ensuring integrity;
- *Block Storage (Cinder)*: The block storage system allows users to create block-level storage devices that can be attached to or detached from VM instances;
- *OpenStack Networking (Neutron)*: Neutron is a system for managing networks and IP addresses. The system allows users to create their own networks and assign IP addresses to VM instances;
- *OpenStack Identity (Keystone)*: Keystone is an account management service that acts as an authentication and access control system;
- *OpenStack Image (Glance)*: It supplies a range of VM image management capabilities from discovery and registration to delivery of services for disk and server images.

To add spot market facilities to OpenStack, we extended Horizon to be capable of using the services provided by SipaaS. Horizon's main panel includes two different sections, one for members with system administration role and another section for other standard users. Since the admin section is only visible to users with administrator privileges, we added a new panel to this section, through which system administrators are capable of enabling spot market support and can define global settings such as maximum and minimum amount of bid price for users, number of available VMs for allocation, and the maximum number of VMs a user can request. These parameters are passed to SipaaS by calling corresponding services defined in SipaaS and discussed in earlier parts of this section.

We also added another panel to the section visible to standard OpenStack users, labeled as Spot Instances. As it can be seen in Figure 7.5, this panel allows users to submit


Request Spot Instances

Availability Zone:	nova
Instance Name: *	test
Flavor: *	m1.nano
Instance Count: *	3
Bid Amount: *	0.0190
Instance Boot Source: *	Boot from image
Image Name:	cirros-0.3.2-x86_64-uec (24.0 MB)

Specify the details for launching an instance.

The chart below shows the resources used by this project in relation to the project's quotas.

Flavor Details

Name	m1.nano
VCPUs	1
Root Disk	0 GB
Ephemeral Disk	0 GB
Total Disk	0 GB
RAM	64 MB

Project Limits

Number of Instances	0 of 20 Used
Number of VCPUs	0 of 20 Used
Total RAM	0 of 51,200 MB Used

Figure 7.5: Screenshot of requesting spot instances web page.

their bids and the number of required spot instances (i.e., orders) to the system. At the backend, when a user submits his request for using spot instances, a unique reference number is created for each order and this reference number with corresponding bid price and number of requested instances are sent to Sipaas by calling its *addorder* service and are stored in a database table named *order*. According to the computed and returned price by SipaaS, if user's bid amount is greater than the price, which means the request can be fulfilled, requested instances are created and two local database tables named *order* and *instance* are updated. Figure 7.6 shows an EERD of the local database including *order* and *instance* tables created in the OpenStack platform.

In the *instance* table, order reference created at the previous step and instance ids created after launching VMs are stored. Later, if a user terminates any of the spot instances, the table is updated accordingly and the *updateorder* service of SipaaS is invoked to calculate the new price. If a user terminates all the instances belonging to a single order,

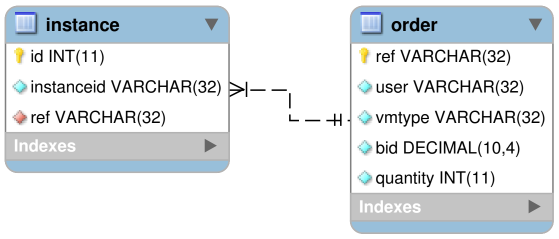


Figure 7.6: EERD of the database used for horizon extensions.

then *removeorder* is called and after both mentioned operations, the instances of other users, which belong to an order with bid price lower than the current market price, are terminated automatically (out-of-bid orders termination). Figure 7.7 shows the sequence diagram of the process of handling an order submission by a user.

There is another added panel labeled spot pricing history, in which users are able to view the history of spot price fluctuations. By supplying the desired duration, users can see the plotted result from the invoked *pricehistory* service of Sipaas. A sample screenshot of the spot pricing history panel is depicted in Figure 7.8.

7.2.3 Pricing Mechanism

This section discusses the implemented algorithm in the pricing module of SipaaS. It is important to note that the presented algorithm is not the main focus of the current chapter, and it can be replaced by any arbitrary pricing mechanism. The pricing mechanism plugged into the SipaaS framework works based on the online Ex-CORE auction algorithm proposed in Chapter 6. Here, we discuss the general concepts of online Ex-CORE auction while details of the proposed auction can be found in Chapter 6.

The Ex-CORE algorithm generates a market-wide single price for each auction round according to the current available orders in the market. The main aim of the Ex-CORE algorithm is to maximize the provider's profit while it is strategy-proof (truthful). An auction mechanism is *strategy-proof* –also called truthful or incentive-compatible– if the *dominant* bidding strategy for every bidder is to always report their true valuation irrespective of the behavior of the other bidders. In Chapter 6, it is shown that the Ex-CORE auction has a high probability of being truthful, and generates a near optimal profit for

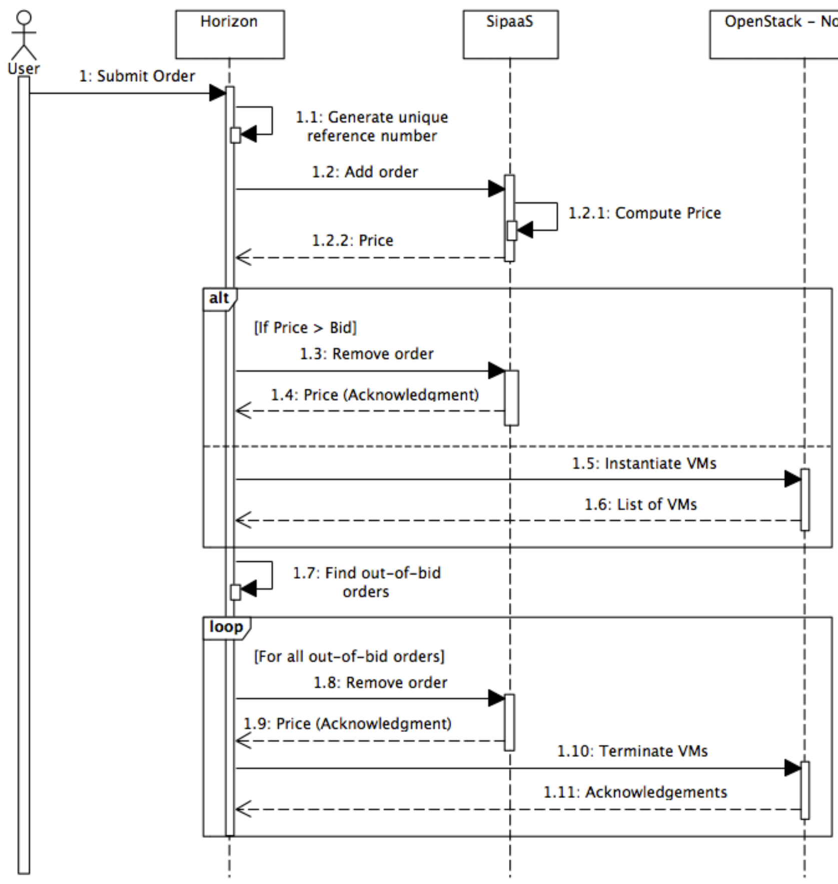


Figure 7.7: Sequence diagram of an order submission handling.

the provider in each round of auction.

The online Ex-CORE auction operates in a greedy manner where it attempts to maximize the revenue given the newly arriving order and the existing orders at a given time. To maximize revenue, the random component of the auction mechanism must generate the price in a way that the gained revenue is a good estimate of the revenue generated by the *Optimal Single Price* auction. The *Optimal Single Price* auction, $\mathcal{F}$, is defined as follows:

Definition 9. *Let* $\mathbf{d}$ *be an order vector. An order,* d_i*, is a pair of* (r_i, b_i)*, where* r_i *represents the number of required items and* b_i *the bid price. Without loss of generality, suppose the components of* $\mathbf{d}$ *are sorted in descending order by bid prices. The auction* $\mathcal{F}$ *on input* $\mathbf{d}$ *determines the value* k *such that* $b_k \sum_{i=1}^{k} r_i$ *is maximized. All bidders with* $b_i \geq b_k$ *win at price* b_k *and all remaining*

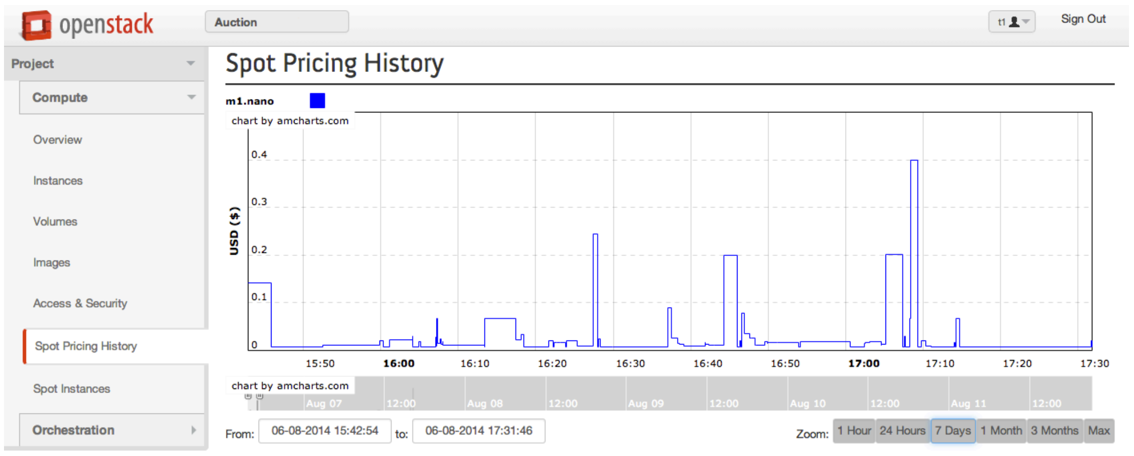


Figure 7.8: Screenshot of spot pricing history web page.

bidders lose. Thus, the revenue of $\mathcal{F}$ on input **d** *is*

$$\mathcal{F}(\mathbf{d}) = \max_i \; b_i \sum_{j=1}^{i} r_j \,. \tag{7.1}$$

We omit details of the online Ex-CORE auction in this chapter and the interested readers are referred to Chapter 6. To retrieve price, web services in SipaaS call the auction algorithm in pricing module. For example, each time *addorder*, *updateorder*, and *removeorder* web services are invoked, the auction algorithm is called to generate the current price based on the newly updated order vector. The online Ex-CORE records the optimal sale price computed by Optimal Single Price auction in each round, and updates the sale price only when the optimal sale price calculated in the current round differs from the one in the previous round of the auction. This prevents the market to be exposed to high price fluctuations due to existing random component in the Ex-CORE algorithm.

7.3 Evaluation and Validation

In this section, we support our proposed framework by conducting an experimental study in a real environment. The goals are two-fold: (i) to demonstrate that the extension software to the Openstack combined with the SipaaS framework can operate in a practical environment to provide spot market facilities and (ii) to evaluate the system's

Table 7.2: Types of VM instances and their specifications used to host system components in the experiment.

Instance Type	Cores	CPU (ECU*)	Memory (GB)	Storage (GB)
`m1.small`	1	1	1.7	160
`m3.2xlarge`	8	26	30	160

* One ECU (EC2 Compute Unit) is equivalent to CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

behavior and our auction pricing model in a test experiment.

7.3.1 Experimental Testbed

The testbed used for the evaluation of the system consists of two main hosts both running Ubuntu 14.04 as operating system. One was used for running SipaaS and the other one used for running all the OpenStack services. The *devstack OpenStack*[3], which is suitable for development and operational testing, is used in the experiment. Hosts used in the experiment are `m1.small` and `m3.2xlarge` VM instances running on Amazon EC2 in `Asia Pacific-Sydney` region. Resource configuration of VM instance types used in the experiment can be seen in Table 7.2.

The `m1.small` machine was chosen to deploy SipaaS Web services on Apache Tomcat 6 web server[4] and MySQL as a DBMS to host and manage the database. The `m3.2xlarge` was used to host all the OpenStack services including Horizon and its extension, deployed on Apache HTTP server Version 2. MySQL was also installed in this machine to host the database for storing all the required data for the extension software.

In the experimental study, users can use their own desktop or labtop PC with a web browser (preferably Google chrome) to connect to the dashboard and use the spot market services.

[3]Deploying OpenStack for Developers, `http://devstack.org/`.
[4]Apache Tomcat, `http://tomcat.apache.org/`

7.3.2 Experimental Design and Setup

We conducted a 20-minute experiment with 10 participants (i.e., spot market users). Participants were divided into two groups of five: (i) Group T or truthful bidders and (ii) Group C or counterpart bidders who have the freedom to misreport their true private values to maximize their utility. Participants of the latter are selected from a group of professional cloud users who have satisfactory level of knowledge about the spot market. Each participant was provided with a user name and password to access the OpenStack dashboard. We provided participants of the experiment with a pair of uniformly random generated quantity and price values that must be considered as their true private values.

Considering the scale of the experiment, we limited the maximum number of simultaneous VM instances each user can run in the system to 2. Accordingly, we chose uniformly random true quantity values from $\{1, 2\}$. For price values, we adopted the pricing details of Amazon EC2 `m3.medium` in the `Asia Pacific-Sydney` region region at the time of the experiment, while we replaced per hour charging period with 30 seconds in our experiment. True price values in dollars are drawn from a uniform distribution of the interval [0.0081, 0.0700], where \$0.0081 and \$0.0700 are the minimum spot instance (reserve price) and the on-demand price per hour for `m3.medium` instances in Amazon, respectively. The maximum possible bid price is also derived from the maximum allowed bid price in Amazon EC2 for the same type of spot instances, i.e., \$0.4520/hour.

Assuming that the provider offers on-demand pricing concurrently and QoS for spot instances are lower than equivalent to on-demand ones, true values higher than on-demand price seems unreasonable. Therefore, we distributed true private price values between the minimum spot price and on-demand price for `m3.medium` instances. However, experiment participants are allowed to submit orders with a bid price higher than the on-demand price. Table 7.3 shows true private price and quantity values for participants of each group.

Similar to Amazon, spot instances are not charged for their partial 30 seconds upon their termination by the provider, while a partial 30 seconds of usage accounted for as a full 30 seconds upon termination by the user. Each full time slot usage (i.e., 30 seconds) is charged based on the spot market price at the beginning of the time slot.

Table 7.3: True private values of experiment participants.

User	Price Value ($)	Quantity
T1, C1	0.0691	2
T2, C2	0.0092	1
T3, C3	0.0475	1
T4, C4	0.0232	2
T5, C5	0.0184	1

Participants of the experiment are provided with the details of the online Ex-CORE auction algorithm and their utility function for one time slot instance usage, formulated as below:

$$u(r,b) = \begin{cases} (qv - rp)x\,, & \text{if } b \geq p \text{ and } r \geq q; \\ 0\,, & \text{otherwise.} \end{cases} \tag{7.2}$$

where r, b, q, v, p, and x are the requested number of instances, bid price value, true private quantity value, true private price value, spot market price at the time of order submission, and a Boolean value describing whether the order is accepted or not, respectively.

Participants are asked to acquire VM instances of type `m1.nano` as long as they can according to their true private values using the OpenStack dashboard. Participants of group T are obliged to submit their true values to acquire instances through the whole experiment regardless of the market price fluctuation. Participants of group C are asked to try to maximize their utility based on rules of the experiment and given pricing information. Therefore, if it is deemed beneficial, a participant of group C might strategically misreport her/his bid price or the quantity, i.e., $b \neq v$ or $r \neq q$. To provide enough incentives for participant of group C to act rationally in the experiment, we considered a prize for the winner of the experiment. The winner of the experiment is the one who can achieve the highest positive difference of the utility value with his/her counterpart truthful bidders.

All participants are taught the goal, rules and details of the experiment. The experiment was conducted in the real environment where participants have been able to run instances according to their order submissions. The results of the experiment are dis-

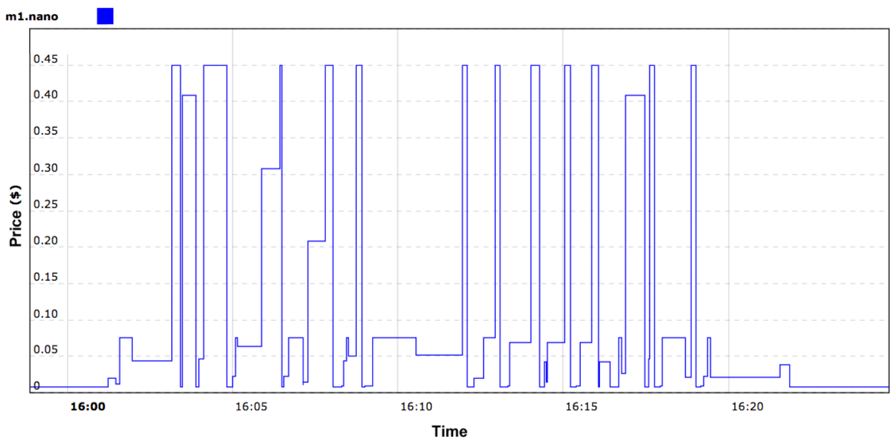


Figure 7.9: Spot market price fluctuation during the experiment.

cussed in the following section.

7.3.3 Results and Analysis

The main goal of the conducted experiment is to show that the system works flawlessly in a practical test environment. All the participants experienced valid system behavior in the experiment. They were able to submit their orders into the system and boot up their instances whenever their bid was higher than the market price. As soon as market price went above of the submitted bid price, acquired instances were terminated by the system immediately without any prior notice.

Figure 7.9 depicts the market price fluctuation during the experiment. As shown in the figure, the price reaches the maximum bid price in multiple cases. This happened due to the reason that some low value participants, for example $C4$ and $C2$, who were starving in the market, tried to terminate other participants' instances by submitting very high bid price and terminating their instances immediately to submit their new orders. This, however, affected their utility value since they were charged multiple times higher than their true values.

In Chapter 6, it is proved that Ex-CORE auction mechanism is truthful with high probability. Therefore, as we expected, excluding $T3$, all truthful users (i.e., participants of group T) achieved higher utility value than their counterpart users who misreported their true values. Table 7.4 shows the total cost and achieved utility values by all users based on the utility function in Equation (7.2).

In order to investigate how user *C*3 could achieve the highest positive difference with his/her paired truthful participant, we analyzed the submitted orders by all users. The result of our analysis shows that *C*3 is the most truthful user among the participants of group *C*, who continuously submitted the true quantity value and bid price values significantly close to his/her true value. The only reason *C*3 achieved highest difference is that she/he was quicker in submitting orders and could obtain two additional full time slots of instance usage while the truthful user *T*3 was also able to do the same with submitting his/her true values at the same time.

Table 7.4: Total cost, the number of full time slots usage, and utility values of experiment participants.

User	Total Cost ($)	Number of Full Time Slots	Utility Value ($)
T1	1.2964	16	**0.9148**
C1	1.8216	17	0.5278
T2	0.0000	0	**0.0000**
C2	10.0227	18	-9.8571
T3	0.1896	6	0.0954
C3	0.2280	8	**0.1520**
T4	0.0436	1	**0.0030**
C4	3.6810	5	-3.4490
T5	0.0000	0	**0.0000**
C5	0.0738	2	-0.0370

As it can be seen in Table 7.4, *T*2 and *T*5 could not acquire instances for a full time slot at all, since the market price was often higher than their true price values. *T*4 similarly ends up running instances for only one time slot. Comparatively, paired users from group *C* acquired instances for higher number of time slots. However, their overall utility values are negative, as they ended up paying more than their true values.

Results of the experiment reported in Table 7.4 supports the theoretically proven supposition that Ex-CORE algorithm is truthful with high probability. This confirms the fact that rational users' dominant strategy in a truthful auction mechanism is to report their true private values. Moreover, our investigation on the historical price data of spot instances in Amazon EC2 shows that price spikes similar to what happened in our experiment are occurring in Amazon's spot market as well. This might happen due to the same experience we had in our experiment where some users submit very high bids or possibly sudden spikes in demand. Intuitively, without knowing how the spot market mechanism works, no user has the incentive to strategize over its bid. This has been suggested by other studies as well [9, 127].

7.4 Summary and Conclusion

In this chapter, we introduced an open source Spot instance pricing as a Service (SipaaS) framework that provides a set of web services to facilitate running a spot market. We also presented our extension software to Horizon – OpenStack dashboard project – that makes use of the SipaaS framework to run a spot market in the OpenStack platform. In order to evaluate and validate our proposed system, we conducted an experimental study with a group of ten participants. The results of the experimental study support the validity of the proposed system and demonstrates the behavior of the system. In addition, our study experimentally confirms the truthfulness of the auction pricing mechanism used in the SipaaS framework. To conclude, those IaaS cloud providers interested to run spot market resembling the Amazon EC2 spot instances could consider our proposed spot instance pricing as a service framework using the Ex-CORE auction algorithm as a relevant replacement. Considering that both are of similar structures, we would expect the same market reaction and similar market pricing behavior.

This page intentionally left blank.

Chapter 8
Conclusions and Future Directions

This chapter summarizes the research work on market and economics-inspired mechanisms for maximizing IaaS cloud providers' profit and highlights the major findings in this thesis. It also discusses future research directions and open research problems in the area.

8.1 Summary of Contributions

CLOUD computing is changing the way industries and enterprises do their businesses by delivering IT as a service. This computing paradigm has been driving the massive migration from single-user, in-house computing servers to multi-tenant, centrally hosted cloud based alternatives. While migration to cloud helps customers to avoid upfront technology investments and offers significant cost benefits, which have been extensively discussed, comparatively lower attention has been devoted to challenges and opportunities of cloud vendor companies in respect to their profitability.

In this regard, the thesis set out with one general goal *to maximize IaaS cloud provider's profit* breaking down into objectives delineated in Chapter 1. In order to achieve these objectives, we proposed and investigated a set of market and economics-inspired mechanisms for IaaS cloud providers, including resource management, financial option market, revenue management, and auction mechanism design. Our proposed mechanisms have been designed for two main scenarios forming two main parts of the thesis: 1) when the provider acts solely using their in-house computing resources to serve customers and 2) when it participates in a cloud federation and benefits from resource sharing among providers.

Chapter 2 presented an in-depth review and analysis of the related work including the background and classification of methods and mechanisms for profit maximization of cloud providers. In addition, it surveyed the relevant aspects that motivate cloud federation and studied economics-related challenges of cloud federation. The literature review has helped us to identify gaps, open challenges, and the research direction of the thesis.

Chapter 3 proposed policies to enhance an IaaS provider's profit when the provider is a member of a cloud federation. The proposed resource provisioning policies have been designed for IaaS cloud providers supporting two different and jointly offered types of QoS and pricing models namely spot and on-demand. This allows to cancel less profitable VMs (i.e., spot VMs) in favor of more profitable requests (i.e., on-demand VMs). Moreover, in order to leverage the cloud federation potentials, we proposed a cloud exchange market and appropriate demand-oriented pricing model for federated cloud environments.

Using our policies, we evaluated the impact of different parameters such as ratio of spot VMs to the total number of VMs, percentage of persistent spot VMs, number of providers in the federation, and provider's load on various performance metrics such as profit, utilization, and rejection rate. Results demonstrated that our policies help providers to increase profit and to reject fewer requests, while keeping utilization at an acceptable level. Experimental results also allowed us to derive some guidelines for IaaS providers. For example, running on-demand requests locally is more profitable if ratio of spot VMs to total number of VMs is high and termination of spot VMs may lead to less discontinuation of the service consumption. Moreover, outsourcing is more profitable when spot VMs are scarce and termination may result in discontinuation of the service consumption.

To address the next objective, Chapter 4 considered a cloud provider offering subscription-based pricing model in addition to on-demand pay-as-you-go pricing model in a federated cloud environment. Customers reserve resources in advance and may or may not fully utilize the resources later on. The main aim of the chapter is to exploit the underutilized reserved capacity to accommodate requests form customers of other pricing channels. Nevertheless, Quality of Service (QoS) to customers who reserve resources in advance should be satisfied. Therefore, a financial option market model for cloud federation was proposed. The proposed model guaranties resources to reserved customers whenever they need them, while keeping resources utilized all the time. This model allows the provider to hedge against the critical and risky situation in which customers request their reserved resources while all the resources have been allocated to other users, by trading (buy or outsource) resources from other service providers in the cloud federation.

Experimental results showed that financial option-based contracts between cloud providers in a cloud federation would help them to exploit the underutilized reserved capacity without any concern to acquire the needed resources at any given time. Using our model, the provider can increase the profit while keeping the rejection rate of reserved requests at a negligible level. The model therefore, contributes to obtaining a trust and goodwill from the provider's client base.

In Chapter 5, we presented a revenue management framework to tackle the problem of optimal capacity control for allocating resources among customers who are segmented into three main markets, i.e., reservation, on-demand pay-as-you, and spot markets. The main challenge is that the provider must determine the optimal capacity to admit demands from the reservation market such that the expected revenue is maximized. We considered the stochastic lifetime of on-demand pay-as-you-go requests and reserved capacity utilization and accordingly we formulated the problem as a finite horizon Markov decision processes. Because finding the optimal solution is computationally prohibitive in practical settings, we presented two algorithms, namely *pseudo optimal* and *heuristic*, that reduce the computational complexity. Large-scale simulations based on Google cluster usage traces with Amazon EC2 pricing were conducted to evaluate the revenue performance of the proposed revenue management framework using our capacity control algorithms. We further evaluated the proposed algorithms with comparison to the optimal algorithm in a small-scale scenario. Our experimental results suggest that significant revenue increase could be expected through the proposed revenue management system given that sufficient resource contention is present in the system.

In Chapter 6, we presented an envy-free auction that is truthful with high probability and generates a near optimal profit for the cloud provider. The auction operates similarly to the AWS EC2 spot market. The truthfulness of the mechanism frees bidders from having to understand its intricacies, thereby lowering the complexity of participation and the options for strategic behavior. At the same time, the mechanism aims to attain a maximal profit for the provider, and achieves the property of being envy-free through the use of a uniform price. In addition, in order to incorporate marginal costs of production in the resource trading process, we pair the auctioning scheme with a method that calculates dynamic reserve prices based on a marginal cost model considering data center *PUE*, load, and electricity cost. An important benefit of the proposed auction design is that it achieves near optimality in the form of maximizing revenue without requiring prior knowledge on the bid distribution.

Our evaluation demonstrated the proposed auction mechanism performance under a variety of order distributions. We showed that the proposed mechanism significantly outperforms the uniform price auction. It also closely approximates the profit outcome of the revenue maximizing, but non-truthful, optimal single price auction in an online setting (within 6% in our experiments), while improving on the number of rejected VMs (up to 17% in our experiments). Finally, our results showed that the generated revenue does not differ significantly from the revenue attained by a benchmark mechanism based on dynamic programming relying on prior knowledge regarding the holding time of VMs.

Finally, Chapter 7 introduced an open source framework called Spot instance pricing as a Service (SipaaS) that is a realization of the proposed auction pricing in Chapter 6. Our prototype system provides a set of web services to facilitate running a spot market in IaaS cloud environments. Extension plugin software for the OpenStack dashboard project was presented to make use of the framework's web services. By using the offered web services and the implemented extension to the dashboard program, one can execute a spot market in the OpenStack platform for selling VM instances based on dynamic form of pricing.

In order to evaluate and validate the system, we conducted an experimental study with a group of ten professional cloud users familiar with the spot market concept. The result of the experimental study demonstrated the validity of the framework and showed that the proposed auction pricing mechanism used in the framework is truthful with high probability. In addition, the built spot market using our framework exhibited an analogous market reaction and market pricing behavior to that of spot instances in the AWS environment.

8.2 Future Research Directions

Few publications have been done about opportunities and challenges regarding the economic structure of cloud service providers as a multidisciplinary area of research shared among disciplines such as economics, business, finance and computer science. Market mechanisms are still in their infancy and pricing models are relatively immature. In addition, technologies and mechanisms to maximize the Return on Investment (ROI) for cloud service providers are yet to be investigated in more detail. In spite of the significant contributions of the current thesis in developing and introducing mechanisms for maximizing profit for IaaS cloud providers, there are many open research challenges that need to be addressed in order to further advance the area. This section outlines several open issues that can serve as a starting point for future research.

8.2.1 Advanced Resource Provisioning Policies

In Chapter 3, we proposed policies that help in the decision-making process to increase utilization and profit of a federated IaaS cloud provider. In our model, providers benefit from outsourcing on-demand pay-as-you-go requests while they also have the option of terminating spot VMs in favor of these requests. Our proposed policies only address outsourcing on-demand requests, not spot requests. Hence, design of market mechanisms for federated cloud environments that allows for outsourcing spot requests with dynamic pricing model is of great value. Decision equations in this case must be designed to handle the highly fluctuating spot VM prices. Dynamic pricing of resources to sell idle capacity of the data center for an individual cloud has been explored in Chapter 6. However design of efficient dynamic pricing models suitable for federated cloud environments has been left as an open issue.

Furthermore, in our decision equations, we did not include the cost savings occurring via unused physical servers shutdown. It is important to design and investigate strategies that include cost of shutting down unused physical servers of the data centers due to reduction of electric power consumption, in addition to termination of spot VMs and outsourcing of on-demand VM requests.

Lastly, proposed resource-provisioning policies are designed to act based on instant time profit calculation without any knowledge about future requests and lifetime of accepted requests. Developing online algorithms that consider the prediction of future demand and resource availability to drive outsourcing decisions can be explored as an extension to our study.

8.2.2 Option Trading Strategies

Chapter 4 of this thesis aimed at designing a financial option market for federated cloud environments. It also proposed basic strategies regarding buying and selling option contracts in the market. Design and development of advanced option trading strategies can be considered as an extension point for the current work.

For example, in real-world option markets, various option contracts can be combined into an option *portfolio* according to the hedging strategy. Thus, depending on the expected volatility in the market, such combinations can lead to a higher value. Furthermore, our proposed strategy needs to buy option contracts if the on-demand request is accommodated in the reserved capacity. However, for providers with a large amount of physical resources, it might be more beneficial to buy option contracts in bulk with longer expiration dates beforehand. This deals with efficient load and price prediction to gain a significant advantage of acquisition of the option contracts.

Another potential improvement is to design efficient strategies for selling option contracts. Terms of option contracts, i.e., the *premium*, the *strike price*, and the *expiration date* must be determined by the seller of the contract in a way that maximizes cloud provider's profit. Moreover, each provider must decide on the number of option contracts to sell. The following research questions in this regard must be rigorously investigated: 1) How many options to sell? 2) What should be the expiration date of option? 3) For which price, i.e., premium and strike, are they sold? Furthermore, in proposed option market, we only explored trading call options. It will be interesting to extend the market for trading put options that will give providers with a large amount of physical resources the right to sell resources at their will.

8.2.3 Game Theoretical Analysis of Cloud Federation

This thesis considered cloud federation (i.e., the ability to acquire third party capacity) as a potential basis of increasing profit. It is conceivable that IaaS providers can obtain significant economic value from access to third party resources. In fact, a federation increases the degree of elasticity for the cloud providers. However, Chapters 3 and 4 only investigate resource sharing from the perspective of an individual cloud provider which is selfishly trying to maximize its own profit. To achieve coordination between providers and to establish a cloud federation, it is important to precisely investigate the interactions among these selfish cloud providers. Economic models and strategies must be developed to regulate the capacity sharing among cloud providers, aimed at maximizing their own profit. These strategies must guarantee the benefits for all cloud providers (Social welfare) and provide enough incentives for each individual to contribute resources in the cloud federation. In this regard, one possible research direction is the development of game theoretical approaches to model cooperation and conflicts in the resource sharing between rational and selfish cloud providers in a cloud federation.

8.2.4 Customer Diversion in Revenue Management Framework

The broad literature of revenue management provides many relevant future directions to the revenue management framework proposed in Chapter 5. The proposed revenue management framework incorporates an admission control to compute the maximum number of reservation contracts the provider can accept. Rejection of a reservation contract (also known as bumping a customer) might result in disappointed customers who diverge from subscription-based pricing model to other pricing models or even similar services from other providers. This customer switching behavior is called *diversion* in the literature and has been investigated in revenue management systems for Airline and Hotel industries. However, more research needs to be done on modeling customer reaction as well as how they switch between markets in cloud computing environments once they are bumped by the admission control mechanism. Optimization algorithms must incorporate the diversion of customers to other pricing models and competitors. In addition, future research needs to be done to integrate our proposed reserved capacity control with support for investment decisions on extending the infrastructure in a practical real-world system to prevent excessive rejection of requests.

8.2.5 Revenue Management with Overbooking

Cloud providers offering the subscription-based pricing model are liable to offer guaranteed availability for these VM requests to honor the associated SLA. In order to guarantee the availability, the proposed revenue management framework does not allocate requests from other pricing channels to underutilized reserved capacity of the data center. Allowing requests from other pricing channels to be accommodated in the reserved capacity of data center might result in capacity *overbooking* and creates a risk of SLA violation. However, such overbooking generates more revenue for the cloud provider. One possible future direction of this work can be the extension of the revenue management framework with support for overbooking techniques.

8.2.6 Multi-dimensional Truthful Mechanism Design

Chapter 6 presented a design and application of a multi-unit, online recurrent auction mechanism within the context of IaaS resource trading. The proposed auction mechanism generates near optimal profit for the provider and with high probability is truthful in the bid price dimension while it is not truthful in the quantity dimension. The design limits the maximum number of instances that a customer can request to certify the truthfulness in bid price dimension. It is also shown through simulation that as the market size grows, the probability of raising utility through misreporting quantity converges to zero. In addition, the proposed mechanism sets the price and allocations in a greedy manner in each auction round. That is, the lifetime (duration) of requests has not been taken into account. The chapter proposed the oracle optimal auction mechanism that knows the lifetime of VM requests in advance to calculate the price and the set of accepted orders. The proposed algorithm however is not truthful and inquires about the duration of requests, which is not common knowledge in cloud marketplaces.

Although designing a multi-dimensional truthful mechanism without a priori knowledge of the order distribution and lifetime of requests might be very complex, a potential future research direction could be the design of mechanisms that are two-dimensionally truthful (i.e., bid price and quantity) and somehow take into account requests' duration.

8.3 Final Remarks

Cloud computing has evolved as a key IT platform that aims at providing computing resources for hosting applications as a utility. Market and economics-inspired mechanisms explored in this thesis will enable cloud providers to increase their return on investment while they honor QoS requirements of customer applications. Cloud computing, as a disruptive technology with great potential to transform IT industry, promises ever-increasing market demands, and consequently more competition prevailing among cloud providers. Research similar to what has been done in this thesis, therefore, becomes more important in driving further innovation and development in cloud computing.

This page intentionally left blank.

Bibliography

[1] V. Abhishek, I. A. Kash, and P. Key, "Fixed and market pricing for cloud services," in *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Orlando, Florida, USA, Mar. 2012, pp. 157–162.

[2] D. Allenotor and R. K. Thulasiram, "Grid resources pricing: A novel financial option based quality of service-profit quasi-static equilibrium model," in *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing*, 29 2008-oct. 1 2008, pp. 75 –84.

[3] B. An, V. Lesser, D. Irwin, and M. Zink, "Automated negotiation with decommitment for dynamic resource allocation in cloud computing," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, Toronto, Canada, May 2010, pp. 981–988.

[4] A. Anandasivam, S. Buschek, and R. Buyya, "A heuristic approach for capacity control in clouds," in *Proceedings of IEEE Conference on Commerce and Enterprise Computing (CEC'09)*, Vienna, Austria, July 2009, pp. 90–97.

[5] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under SLA constraints," in *Proceedings of the IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS'10)*. Miami: IEEE, Aug. 2010, pp. 257 –266.

[6] T. Aoyama and H. Sakai, "Inter-cloud computing," *Business & Information Systems Engineering*, vol. 3, pp. 173–177, 2011.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[8] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366–1379, July 2013.

[9] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing," *ACM Transaction Economy Computing*, vol. 1, no. 3, pp. 16:1–16:20, Sept. 2013.

[10] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the InterCloud - protocols and formats for cloud computing interoperability," in *Proceedings of the 4th International Conference on Internet and Web Applications and Services*, Venice, Italy, May 2009, pp. 328–336.

[11] D. Bernstein, D. Vij, and S. Diamond, "An intercloud cloud computing economy - technology, governance, and market blueprints," in *Proceedings of 2011 Annual SRII Global Conference (SRII)*, San Jose, California, USA, Apr. 2011, pp. 293 –299.

[12] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, pp. 637–654, Jan 1973.

[13] A. Bossenbroek, A. Tirado-Ramos, and P. M. A. Sloot, "Grid resource allocation by means of option contracts," *Systems Journal, IEEE*, vol. 3, no. 1, pp. 49–64, March 2009.

[14] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10)*, vol. 6081, Busan, South Korea, May 2010, pp. 13–31.

[15] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[17] R. N. Calheiros, A. N. Toosi, C. Vecchiola, and R. Buyya, "A coordinator for scaling elastic applications across multiple clouds," *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1350–1362, Mar. 2012.

[18] G. S. Campbell and J. M. Norman, *Introduction to Environmental Biophysics*. Springer Verlag, 1998.

[19] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to enhance cloud architectures to enable cross-federation," in *Proceedings of the 3rd International Conference on Cloud Computing (Cloud'10)*, Miami, USA, Jul. 2010, pp. 337–345.

[20] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proceedings of IEEE Asia-Pacific Services Computing Conference (APSCC'09)*. Jeju Island: IEEE, Dec. 2009, pp. 103–110.

[21] ——, "Optimization of resource provisioning cost in cloud computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, April 2012.

[22] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. USENIX Association, 2010.

[23] J. C. Cox, S. A. Ross, and M. Rubinstein, "Options pricing: a simplified approach," *Journal of Financial Economics*, vol. 7, pp. 229–263, 1979.

[24] D. Crockford, "The application/json media type for javascript object notation (JSON)," 2006.

[25] A. Danak and S. Mannor, "Resource allocation with supply adjustment in distributed computing systems," in *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS'10)*, Genoa, Italy, Jun. 2010, pp. 498–506.

[26] M. D. de Assunção, A. di Costanzo, and R. Buyya, "A cost-benefit analysis of using cloud computing to extend the capacity of clusters," *Cluster Computing*, vol. 13, no. 3, pp. 335–347, Sep. 2010.

[27] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (Cloud'10)*. Los Alamitos: IEEE Computer Society, Jul. 2010, pp. 228–235.

[28] E. Elmroth, F. G. Marquez, D. Henriksson, and D. P. Ferrera, "Accounting and billing for federated cloud infrastructures," in *Proceedings of the Eighth International Conference on Grid and Cooperative Computing (GCC'09)*, Lanzhou, China, Aug. 2009, pp. 268–275.

[29] P. Faratin, C. Sierra, and N. R. Jennings, "Negotiation decision functions for autonomous agents," *Robotics and Autonomous Systems*, vol. 24, no. 3, pp. 159–182, 1998.

[30] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[31] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905 – 2922, 2009, virtualized Data Centers.

[32] A. Gohad, N. C. Narendra, and P. Ramachandran, "Cloud pricing models: A survey and position paper." in *Proceedings of IEEE International Conference on cloud Computing in Emerging Markets (CCEM'13)*, Oct 2013, pp. 1–8.

[33] Í. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit," in *Proceedings of the 3rd International Conference on Cloud Computing*. Miami: IEEE Computer Society, Jul. 2010, pp. 123–130.

[34] Í. Goiri, F. Julià, J. Fitó, M. Macías, and J. Guitart, "Resource-level QoS metric for CPU-based guarantees in cloud providers," in *Economics of Grids, Clouds, Systems, and Services*, ser. Lecture Notes in Computer Science, J. Altmann and O. Rana, Eds. Springer Berlin / Heidelberg, 2010, vol. 6296, pp. 34–47.

[35] Í. Goiri, J. Guitart, and J. Torres, "Economic model of a cloud provider operating in a federated cloud," *Information Systems Frontiers*, vol. 14, no. 4, pp. 827–843, 2011.

[36] Í. Goiri, K. Le, J. Guitart, J. Torres, and R. Bianchini, "Intelligent placement of datacenters for internet services," in *Proceedings of the 31st IEEE International Conference on Distributed Computing Systems (ICDCS'11)*, Minneapolis, Minnesota, USA, Jun. 2011, pp. 131–142.

[37] A. V. Goldberg and J. D. Hartline, "Competitiveness via consensus," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA'03)*, Baltimore, Maryland, USA, Jan. 2003, pp. 215–222.

[38] ——, "Envy-free auctions for digital goods," in *Proceedings of the 4th ACM conference on Electronic Commerce (EC'03)*, San Diego, CA, USA, Jun. 2003, pp. 29–35.

[39] A. V. Goldberg, J. D. Hartline, A. R. Karlin, M. Saks, and A. Wright, "Competitive auctions," *Games and Economic Behavior*, vol. 55, no. 2, pp. 242 – 269, 2006.

[40] E. R. Gomes, Q. B. Vo, and R. Kowalczyk, "Pure exchange markets for resource sharing in federated clouds," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 9, pp. 977–991, 2012.

[41] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Computing Communication Review*, vol. 39, no. 1, pp. 68–73, 2008.

[42] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myat, "Best practices for data centers: Lessons learned from benchmarking 22 data centers," *ACEEE Summer Study on Energy Efficiency in Buildings in Asilomar, CA.*, vol. 3, pp. 76–87, 2006.

[43] P. Harsh, Y. Jegou, R. Cascella, and C. Morin, "Contrail virtual execution platform challenges in being part of a cloud federation," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssire, Eds. Springer Berlin Heidelberg, 2011, vol. 6994, pp. 50–61.

[44] M. M. Hassan, B. Song, and E.-N. Huh, "Distributed resource allocation games in horizontal dynamic cloud federation platform," in *Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC'11)*, Banff, Canada, Sep. 2011, pp. 822–827.

[45] Y.-J. Hong, J. Xue, and M. Thottethodi, "Dynamic server provisioning to minimize cost in an iaas cloud," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. San Jose, California, USA: ACM, 2011, pp. 147–148.

[46] J. Hull, *Options, futures and other derivatives*. Pearson Prentice Hall, 2009.

[47] B. Javadi, R. K. Thulasiram, and R. Buyya, "Statistical modeling of spot instance prices in public cloud environments," in *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC'11)*, Melbourne, Dec. 2011, pp. 219–228.

[48] R. Johnson, J. Hoeller, A. Arendsen, and R. Thomas, *Professional Java Development with the Spring Framework*. John Wiley & Sons, 2009.

[49] V. Kantere, D. Dash, G. Francois, S. Kyriakopoulou, and A. Ailamaki, "Optimal service pricing for a cloud cache," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1345–1358, Sept. 2011.

[50] M. M. Kashef, A. Uzbekov, J. Altmann, and M. Hovestadt, "Comparison of two yield management strategies for cloud service providers," in *Grid and Pervasive Computing*, ser. Lecture Notes in Computer Science, J. Park, H. Arabnia, C. Kim, W. Shi, and J.-M. Gil, Eds. Springer Berlin Heidelberg, 2013, vol. 7861, pp. 170–180.

[51] K. Keahey, M. Tsugawa, A. Matsunaga, and J. A. B. Fortes, "Sky computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43–51, 2009.

[52] H. Kim, Y. el Khamra, S. Jha, and M. Parashar, "Exploring application and infrastructure adaptation on hybrid Grid-Cloud infrastructure," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. Chicago: ACM, June 2010, pp. 402–412.

[53] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, "Reducing electricity cost through virtual machine placement in high performance computing clouds," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, Seattle, USA, Nov. 2011, pp. 22:1–22:12.

[54] J.-S. Lee and B. Szymanski, "A novel auction mechanism for selling time-sensitive e-services," in *Proceedings of Seventh IEEE International Conference on E-Commerce Technology, (CEC'05)*, Hong Kong, Jul. 2005, pp. 75–82.

[55] Y. Lee, C. Wang, J. Taheri, A. Zomaya, and B. Zhou, "On the effect of using third-party clouds for maximizing profit," in *Algorithms and Architectures for Parallel Processing*, ser. Lecture Notes in Computer Science, C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo, Eds. Springer Berlin / Heidelberg, 2010, vol. 6081, pp. 381–390.

[56] Y. C. Lee and A. Zomaya, "Rescheduling for reliable job completion with the support of clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1192–1199, Oct. 2010.

[57] H. Li, C. Wu, Z. Li, and F. C. M. Lau, "Profit-maximizing virtual machine trading in a federation of selfish clouds," in *Proceedings of International Conference on Computer Communications (INFOCOM'13)*, Turin, Italy, April 2013, pp. 25–29.

[58] K. Lu, T. Roblitz, R. Yahyapour, E. Yaqub, and C. Kotsokalis, "QoS-aware SLA-based advanced reservation of infrastructure as a service," in *Proceedings of Third IEEE International Conference on Cloud Computing Technology and Science (CloudCom'11),*, Athens, Greece, Nov 2011, pp. 288–295.

[59] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105 – 1122, 2003.

[60] M. Macías, J. O. Fitó, and J. Guitart, "Rule-based SLA management for revenue maximisation in cloud computing markets," in *Proceedings of International Conference on Network and Service Management (CNSM'10)*, Niagara Falls, Canada, Oct 2010, pp. 354–357.

[61] M. Macías and J. Guitart, "A genetic model for pricing in cloud computing markets," in *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC'11)*, Taichung, Taiwan, Mar. 2011, pp. 113–118.

[62] H. Markowitz, "Portfolio selection," *The journal of finance*, vol. 7, no. 1, pp. 77–91, 1952.

[63] M. Mazzucco and M. Dumas, "Reserved or On-demand instances? a revenue maximization model for cloud providers," in *Proceedings of 4th IEEE International Conference on Cloud Computing (CLOUD'11)*. Washington: IEEE, Jul. 2011, pp. 428–435.

[64] T. Meinl, A. Anandasivam, and M. Tatsubori, "Enabling cloud service reservation with derivatives and yield management," in *Proceedings of 12th IEEE Conference on Commerce and Enterprise Computing (CEC'10)*, Nov 2010, pp. 150–155.

[65] T. Meinl and D. Neumann, "A real options model for risk hedging in Grid computing scenarios," in *Proceedings of the 42nd Hawaii International Conference on System Sciences, 2009 (HICSS'09)*. Hawaii: IEEE, Jan. 2009, pp. 1–10.

[66] I. Menache, A. Ozdaglar, and N. Shimkin, "Socially optimal pricing of cloud computing resources," in *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'11)*. Paris, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011, pp. 322–331.

[67] R. C. Merton, "Theory of rational option pricing," *Bell Journal of Economics*, vol. 4, pp. 141–183, 1973.

[68] W. Michalk, L. Lilia Filipova-Neumann, B. Blau, and C. Weinhardt, "Reducing Risk or Increasing Profit? Provider Decisions in Agreement Networks," *Service Science*, vol. 3, no. 3, pp. 206–222, 2011.

[69] M. Mihailescu and Y.-M. Teo, "The impact of user rationality in federated clouds," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*, Ottawa, Canada, May 2012, pp. 620–627.

[70] M. Mihailescu and Y. Teo, "A distributed market framework for large-scale resource sharing," in *Euro-Par 2010 - Parallel Processing*, ser. Lecture Notes in Computer Science, P. DAmbra, M. Guarracino, and D. Talia, Eds. Springer Berlin Heidelberg, 2010, vol. 6271, pp. 418–430.

[71] M. Mihailescu and Y. M. Teo, "Dynamic resource pricing on federated clouds," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*, Melbourne, Australia, May 2010, pp. 513 –517.

[72] ——, "On economic and computational-efficient resource pricing in large distributed systems," in *Proceedings of the 10th IEEE/ACM International Conference on-Cluster, Cloud and Grid Computing (CCGrid'10)*, Melbourne, Australia, May 2010, pp. 838–843.

[73] ——, "Strategy-proof dynamic resource pricing of multiple resource types on federated clouds," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'10)*, ser. Lecture Notes in Computer Science, vol. 6081. Busan: Springer, May 2010, pp. 337–350.

[74] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-placement algorithms for on-demand clouds," in *Proceedings of Third International Conference on Cloud Computing Technology and Science (CloudCom'12)*, Taipei, Taiwan, Dec. 2011, pp. 91–98.

[75] S. V. Mohammadi, S. Kounev, A. Juan-Verdejo, and B. Surajbali, "Soft Reservations: Uncertainty-Aware Resource Reservations in IaaS Environments," in *Proceedings of the 3rd International Symposium on Business Modeling and Software Design (BMSD'13)*, Noordwijkerhout, The Netherlands, July 2013.

[76] H. Moulin and S. Shenker, "Strategyproof sharing of submodular costs: budget balance versus efficiency," *Economic Theory*, vol. 18, no. 3, pp. 511–533, 2001.

[77] R. B. Myerson, "Optimal auction design," *Mathematics of operations research*, vol. 6, no. 1, pp. 58–73, 1981.

[78] S. K. Nair and R. Bapna, "An application of yield management for internet service providers," *Naval Research Logistics (NRL)*, vol. 48, no. 5, pp. 348–362, 2001.

[79] C. Negru and V. Cristea, "Cost modelspillars for efficient cloud computing: position paper," *International Journal of Intelligent Systems Technologies and Applications*, vol. 12, no. 1, pp. 28–38, Jan. 2013.

[80] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

[81] D. Niu, C. Feng, and B. Li, "Pricing cloud bandwidth reservations under demand uncertainty," *SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 151–162, June 2012.

[82] D. Niyato, S. Chaisiri, and B.-S. Lee, "Economic analysis of resource market in cloud computing environment," in *Proceedings of IEEE Asia-Pacific Services Computing Conference (APSCC'09)*, Biopolis, Singapore, Dec. 2009, pp. 156–162.

[83] D. Niyato, A. V. Vasilakos, and Z. Kun, "Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach," in *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*. Newport Beach: IEEE, May 2011, pp. 215–224.

[84] R. S. Padilla, S. K. Milton, and L. W. Johnson, "Service value in IT outsourcing," *International Journal of Engineering and Management Sciences*, vol. 4, no. 3, pp. 285–302, 2013.

[85] R. Pal and P. Hui, "Economic models for cloud service markets," in *Distributed Computing and Networking*, ser. Lecture Notes in Computer Science, L. Bononi, A. Datta, S. Devismes, and A. Misra, Eds. Springer Berlin Heidelberg, 2012, vol. 7129, pp. 382–396.

[86] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot, "Long-term forecasting of internet backbone traffic: observations and initial models," in *Proceedings of Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '03)*, vol. 2, San Francisco, California, USA, March 2003, pp. 1178–1188.

[87] S. Parsons, J. A. Rodriguez-Aguilar, and M. Klein, "Auctions and bidding: A guide for computer scientists," *ACM Computing Survey*, vol. 43, no. 2, pp. 10:1–10:59, Feb. 2011.

[88] M. K. Patterson, "The effect of data center temperature on energy efficiency," in *Proceedings of 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM'08)*, Orlando, Florida, USA, May 2008, pp. 1167–1174.

[89] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Workshop on Energy-Efficient Design (WEED'09)*, 2009.

[90] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[91] A. S. Prasad and S. Rao, "A mechanism design approach to resource procurement in cloud computing," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 17–30, Jan 2014.

[92] T. Püschel, A. Anandasivam, S. Buschek, and D. Neumann, "Making money with clouds: Revenue optimization through automated policy decisions." in *Proceedings of the 17th European Conference on Information Systems (ECIS'09)*, 2009, pp. 2303–2314.

[93] T. Püschel and D. Neumann, "Management of cloud infastructures: Policy-based revenue optimization," in *Proceedings of International Conference on Information Systems (ICIS'09)*, Phoenix, Arizona, Dec. 2009, pp. 2303–2314.

[94] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[95] S. Rahmail, I. Shiller, and R. K. Thulasiram, "Different estimators of the underlying asset's volatility and option pricing errors: parallel Monte Carlo simulation," in *Proceedings of International Conference on Computational Finance and its Applications*, Bologna, April 2004, pp. 121–131.

[96] M. R. Rahman, Y. Lu, and I. Gupta, "Risk aware resource allocation for clouds," Department of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep., 2011.

[97] N. Rasmussen, "Electrical efficiency measurement for data centers," *White Paper by Schneider Electric - Data Center Science Center*, vol. 154 revision 2, 2011.

[98] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov 2011, [Online]. Available: http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

[99] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 1–11, 2009.

[100] B. Rochwerger *et al.*, "Reservoir—when one cloud is not enough," *Computer*, vol. 44, no. 3, pp. 44–51, March 2011.

[101] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1226–1240, Oct. 2010.

[102] H. Roh, C. Jung, W. Lee, and D.-Z. Du, "Resource pricing game in geo-distributed clouds," in *Proceedings of International Conference on Computer Communications (INFOCOM'13)*, Turin, Italy, April 2013, pp. 1519–1527.

[103] M. A. Salehi, B. Javadi, and R. Buyya, "QoS and preemption aware scheduling in federated and virtualized grid computing environments," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 72, no. 2, pp. 231–245, 2012.

[104] N. Samaan, "A novel economic sharing model in a federation of selfish cloud providers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 12–21, Jan. 2014.

[105] P. Samimi, Y. Teimouri, and M. Mukhtar, "A combinatorial double auction resource allocation model in cloud computing," *Information Sciences*, 2014.

[106] L. Schubert, K. Jeffery, and B. Neidecker-Lutz, "The future for cloud computing: Opportunities for european cloud computing beyond 2010," Tech. Rep., 2010. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[107] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya, "Pricing cloud compute commodities: A novel financial economic model," in *Proceedings*

of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12), Ottawa, Canada, May 2012, pp. 451–457.

[108] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings of 31st International Conference on Distributed Computing Systems (ICDCS'11)*, Minneapolis, U.S.A, June 2011, pp. 559–570.

[109] W. F. Sharpe, "Capital asset prices: A theory of market equilibrium under conditions of risk," *Finance*, vol. XIX, no. 3, pp. 425–442, Sep. 1964.

[110] K.-M. Sim, "Grid resource negotiation: Survey and new directions," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, no. 3, pp. 245–257, May 2010.

[111] S. Son and K.-M. Sim, "A price- and-time-slot-negotiation mechanism for cloud service reservations," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 713–728, June 2012.

[112] B. Song, M. M. Hassan, and E.-N. Huh, "A novel cloud market infrastructure for trading service," in *Proceedings of the 9th International Conference on Computational Science and Its Applications (ICCSA)*. Suwon: IEEE Computer Society, Jun. 2009, pp. 44–50.

[113] K. Song, Y. Yao, and L. Golubchik, "Exploring the profit-reliability trade-off in amazon's spot instance market: A better pricing mechanism," in *Proceedings of 21st IEEE/ACM International Symposium on Quality of Service (IWQoS'13)*, Montreal, Canada, June 2013, pp. 1–10.

[114] Y. Song, M. Zafer, and K.-W. Lee, "Optimal bidding in spot instance market," in *Proceedings of the 31st International Conference on Computer Communications (INFOCOM'12)*, Orlando, Florida, USA, Mar. 2012, pp. 190–198.

[115] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a market economy to provision compute resources across planet-wide clusters," in *Proceedings of IEEE*

International Symposium on Parallel Distributed Processing (IPDPS'09), Rome, Italy, May 2009, pp. 1–8.

[116] A. Sulistio, K. H. Kim, and R. Buyya, "Using revenue management to determine pricing of reservations," in *IEEE International Conference on e-Science and Grid Computing*, Bangalore, India, Dec 2007, pp. 396–405.

[117] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: The economics of cloud computing," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'11)*, Portland, Oregon, USA, June 2011.

[118] T. Truong-Huu and C.-K. Tham, "A novel model for competition and cooperation among cloud providers," *IEEE Transactions on Cloud Computing*, vol. 99, no. PrePrints, p. 1, 2014.

[119] K. Vanmechelen, W. Depoorter, and J. Broeckhove, "Combining futures and spot markets: A hybrid market approach to economic Grid resource management," *Journal of Grid Computing*, vol. 9, no. 1, pp. 81–94, Mar. 2011.

[120] J. Varia, "Best practices in architecting cloud applications in the AWS cloud," in *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds. Wiley Press, 2011, ch. 18, pp. 459–490.

[121] K. Vermeersch, "A broker for cost-efficient QoS aware resource allocation in ec2," Master's thesis, University of Antwerp, 2011.

[122] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *The Journal of finance*, vol. 16, no. 1, pp. 8–37, 1961.

[123] W. Voorsluys, J. Broberg, and R. Buyya, *Introduction to Cloud Computing*, R. Buyya, J. Broberg, and A. Goscinski, Eds. John Wiley & Sons, Inc., 2011.

[124] W. Voorsluys and R. Buyya, "Reliable provisioning of spot instances for compute-intensive applications," in *Proceedings of 26th International Conference on Advanced Information Networking and Applications (AINA'12)*, Fukuoka, Japan, Mar. 2012, pp. 542–549.

[125] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, Boston, Massachusetts, USA, June 2010.

[126] P. Wang, Y. Qi, D. Hui, L. Rao, and X. Liu, "Present or future: Optimal pricing for spot instances," in *Proceedings of 33rd IEEE International Conference on Distributed Computing Systems (ICDCS'13)*, Philadelphia, USA, Jul. 2013, pp. 410–419.

[127] W. Wang, B. Li, and B. Liang, "Towards optimal capacity segmentation with hybrid cloud pricing," in *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems (ICDCS'12)*, Macau, China, Jun. 2012, pp. 425–434.

[128] W. Wang, B. Liang, and B. Li, "Revenue maximization with dynamic auctions in IaaS cloud markets," in *Proceedings of the 21st IEEE/ACM International Symposium on Quality of Service (IWQoS'13)*, 2013, pp. 1–6.

[129] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proceedings of IEEE 33rd International Conference on Distributed Computing Systems (ICDCS'13)*, Philadelphia, Pennsylvania, US, July 2013, pp. 400–409.

[130] X. Wang, Y. Xue, L. Fan, R. Wang, and Z. Du, "Research on adaptive QoS-aware resource reservation management in cloud service environments," in *Proceedings of IEEE Asia-Pacific Services Computing Conference (APSCC'11)*, Jeju, Korea, Dec. 2011, pp. 147–152.

[131] X. W. Wang, X. Y. Wang, and M. Huang, "A resource allocation method based on the limited english combinatorial auction under cloud computing environment," in *Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'12)*, May 2012, pp. 905–909.

[132] L. R. Weatherford and S. E. Bodily, "A taxonomy and research overview of perishable-asset revenue management: Yield management, overbooking, and pricing," *Operations Research*, vol. 40, no. 5, pp. 831–844, 1992.

[133] H. Xu and B. Li, "A study of pricing for cloud resources," *SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 3–12, Apr. 2013.

[134] ——, "Dynamic cloud pricing for revenue maximization," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 158–171, July 2013.

[135] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of Spot instances via checkpointing in the Amazon Elastic Compute Cloud," in *In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (Cloud '10)*, Washington, USA, 2010, pp. 236–243.

[136] D. Yoo and K.-M. Sim, "A multilateral negotiation model for cloud service market," in *Grid and Distributed Computing, Control and Automation*, ser. Communications in Computer and Information Science, T.-h. Kim, S. S. Yau, O. Gervasi, B.-H. Kang, A. Stoica, and D. Ślęzak, Eds. Springer Berlin Heidelberg, 2010, vol. 121, pp. 54–63.

[137] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," *Journal of Parallel and Distributed Computing*, vol. 73, no. 4, pp. 495–508, 2013.

[138] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proceedings of IEEE INFOCOM*, 2014.

[139] Q. Zhang, Q. Zhu, and R. Boutaba, "Dynamic resource allocation for spot markets in cloud computing environments," in *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC'11)*, Melbourne, Australia, Dec. 2011, pp. 178–185.

[140] Z. Zhang and X. Zhang, "An economic model for the evaluation of the economic value of cloud computing federation," in *Future Communication, Computing, Control and Management*, ser. Lecture Notes in Electrical Engineering, Y. Zhang, Ed. Springer Berlin Heidelberg, 2012, vol. 141, pp. 571–577.

[141] X. Zheng, P. Martin, and K. Brohman, "Cloud service negotiation: A research roadmap," in *Proceedings of IEEE International Conference on Services Computing (SCC'13)*, Santa Clara Marriott, California, USA, June 2013, pp. 627–634.