SABANCI UNIVERSITY

A Simulation Analysis of Different Allocation and Pricing Policies for Cloud Computing Service Providers

by

Seyed Mohamad Reza Afghah

Supervisor:

Tonguç Ünlüyurt

in the

Faculty of Engineeirng and Natural Sciences

Industrial Engineering

August 2017

Declaration of Authorship

I, Seyed Mohamad Reza Afghah, declare that this thesis titled, 'A Simulation Analysis of Different Allocation and Pricing Policies for Cloud Computing Service Providers' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Seyed Mohamad Reza Afghah

Date: 01/08/2017

A Simulation Analysis of Different Allocation and Pricing Policies for Cloud Computing Service Providers

APPROVED BY:

Assoc. Prof. Dr. Tonguç Ünlüyurt.....
(Thesis Supervisor)

Prof. Dr. Özgür Özlük.....

Assoc. Prof. Dr. Barış Selçuk.....

DATE OF APPROVAL: 01/08/2017

SABANCI UNIVERSITY

Abstract

Faculty of Engineeirng and Natural Sciences
Industrial Engineering

Master of Science

by Seyed Mohamad Reza Afghah

Keywords: Revenue Management; Cloud Computing; Data Centers; Simulation; Pricing Policy

The cloud computing is regarded as a paradigm shift in nowadays IT world. As services of cloud computing behave like perishable products, revenue management techniques can be applied to increase cloud service provider's total revenue. In this thesis, we develop various methods for pricing and capacity allocation. We consider three types of instances; subscription, on-demand and spot instances. We introduce three allocation and pricing policies and propose 8 models based on their combinations. First, we establish a queuing mechanism for on-demand instances which are rejected initially by the cloud with a price incentive. Second, we consider an auction based model for spot instances and introduce two types of threshold policies where it is constant or dependent on the remaining capacity. Finally, the criterion for spot instances selection is based on expected revenue or bid of that customer. We simulate these models on several datasets and evaluate the models with different capacities. The results we obtain indicate the sensitivity of revenue based on the policies we propose over the studied datasets.

SABANCI ÜNIVERSITESİ

Başlık

Bulut bilişim servis sağlayıcıları için farklı kapasite ataması ve fiyatlandırma politikalarının benzetim yoluyla karşılaştırması

Özet

Endüstri Mühendisliği

Anahtar Kelimeler: Gelir yönetimi; bulut bilişim; veri merkezleri; benzetim; fiyatlandırma politikaları

Bulut bilişim, günümüzde bilişim dünyasında paradigma kayması olarak görülüyor. Bulut bilgi işlem hizmetleri kapasite kullanılmadığı zaman bozulabilir ürünlerde olduğu gibi gelir kaybına sebep olurlar. Bu açıdan bakıldığında bulut hizmet sağlayıcısının toplam gelirini artırmak için gelir yönetimi teknikleri uygulanabilir. Bu tezde, fiyatlama ve kapasite tahsisi için çeşitli yöntemler geliştiriyoruz. Üç farklı tip durumu ele almaktayız; Abonelik, talep üzerine ve spot olarak gelen durumlar. Üç farklı tahsis ve fiyatlandırma politikası öneriyor ve bunların farklı kombinasyonlarına dayanarak 8 model öneriyoruz. Öncelikle, isteğe bağlı örnekler için başlangıçta fiyat teşviki içeren bulut tarafından reddedilen bir kuyruklama mekanizması oluşturduk. İkinci olarak, spot örnekler için açık arttırmaya dayalı bir model önerdik ve sabit ya da kalan kapasiteye bağlı iki tür eşik politikası sunduk. Son olarak, spot örneklerin seçimi için kriter olarak, o müşterinin beklenen gelirine veya teklifine dayandığı iki durumu ele aldık. Bu modelleri çeşitli veri setleri üzerinde benzetim yolu ile toplam gelire göre karşılaştırdık. Elde ettiğimiz sonuçlar, incelenen veri kümeleri üzerinde öne sürdüğümüz politikalara dayalı gelir hassasiyetini göstermektedir.

Acknowledgements

Firrst my gratitude goes to my thesis advisor Prof. Dr. Tonguç Ünlüyurt of the Faculty of Engineering at Sabanci Univeristy. The door to Prof. Ünlüyurt office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he deemed I needed it.

I would also like to acknowledge Prof. Dr. Özgür Özlük of the Industrial Engineering Department at MEF Univeristy as the second reader of this thesis, and I would like to send my gratitude to him for his valuable comments and suggestions for this thesis.

Finally, I must express my profound thanks to my parents and to my brother and sisters for supporting me and their continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

Author
Seyed Mohamad Reza Afghah

Contents

List of Figures

List of Tables

Abbreviations

CSP	**C**loud **S**ervice **P**rovider
SLA	**S**ervice **L**evel **A**greement
QoS	**Q**uality **o**f **S**ervice
SaaS	**S**oftware **a**s **a** **S**ervice
PaaS	**P**latform **a**s **a** **S**ervice
IaaS	**I**nfrastructure **a**s **a** **S**ervice
API	**A**pplication **P**rogramming **I**nterface
VM	**V**irtual **M**achine
SipaaS	**S**pot **i**nstance **p**ricing **a**s **a** **S**ervice

Chapter 1

Introduction

Cloud computing, often referred to as simply "the cloud," is the delivery of computing resources (such as CPU, memory, applications, and ...) from data centers to users over the Internet on a pay-for-use basis. Cloud computing provides a platform, where resources such as CPU, software, memory, and information is provided as a service to end-users. The cloud delivers a broad practical, flexible, and cost-effective system for businesses that rely IT needs [2]. This has resulted in a rapid growth for the business of cloud computing, that led to many interesting research questions that can be tackled by operations research techniques. The cloud services are offered by different large scale data centers at a proportionately low cost in the ever changing IT world. In this work, we have scrutinized some features of cloud providers in order to increase their revenue.

"Cloud computing has drawn many researchers' attention to its applications in the IT world as a paradigm shift." Many research have been conducted in the particular case of revenue improvement by different tools, such as McGill & Van Ryzin, and Xu *et al.* [3, 4]. To better understand the dynamics of cloud servers, we will provide a description of the data centers and different services which they offer. In the following we will provide the background information required for understanding this study.

1.1 Data Center

It is the core of a company and the place where the critical processes of the business are run and its information is stored. The technical definition of a data center is a "facility composed of networked computers and storage that businesses or other organizations use to organize, process, store and disseminate large amounts of data" [5]. Large-scale computer systems have been established in the IT world for a while now, and the term data center is widely known to people around the world. In the 1940s, computers were so large that individual rooms had to be specially set aside to house them. Even the steady miniaturization of the computer did not initially change this arrangement because the functional scope increased to such an extent that the systems still required the same amount of space [6]. Even nowadays, with individual PCs being much more powerful than any data processor from those days, every large-scale operation has compound IT infrastructures with a substantial amount of equipment, and they are still contained in well-established chambers. Depending on their size, these are referred to as "server rooms" or "data centers". A server room is referred to a room where it is used to store servers, while a data center is where a whole building is associated with servers.

Data centers are regularly administered by large companies (e.g. Amazon, Google, and ...) or government agencies. However, they are also progressively used to provide and adapt to a fast-growing cloud service environment for private and business applications. This shows the extensive use and necessity for data centers in every aspect of the IT world.

1.2 Data Center Services

The term service provider introduces an establishment which maintains a large scale data center and provides multiple categories of services. There are

three main services offered by these companies: SaaS[1], PaaS[2], and IaaS[3]; where each of them are indicating different layers of a data center's service. Figure 1.1, illustrates the different layers of data centers. In general, the cloud computing is an extensive concept, and it covers just about every possible class of online service. Nevertheless, when businesses refer to cloud procurement, there are usually three models of cloud services under consideration, Software as a Service, Platform as a Service, and Infrastructure as a Service. Each has its own intricacies and hybrid cloud models, in the following we're going to develop an understanding of the high-level differences between SaaS, PaaS, and IaaS.

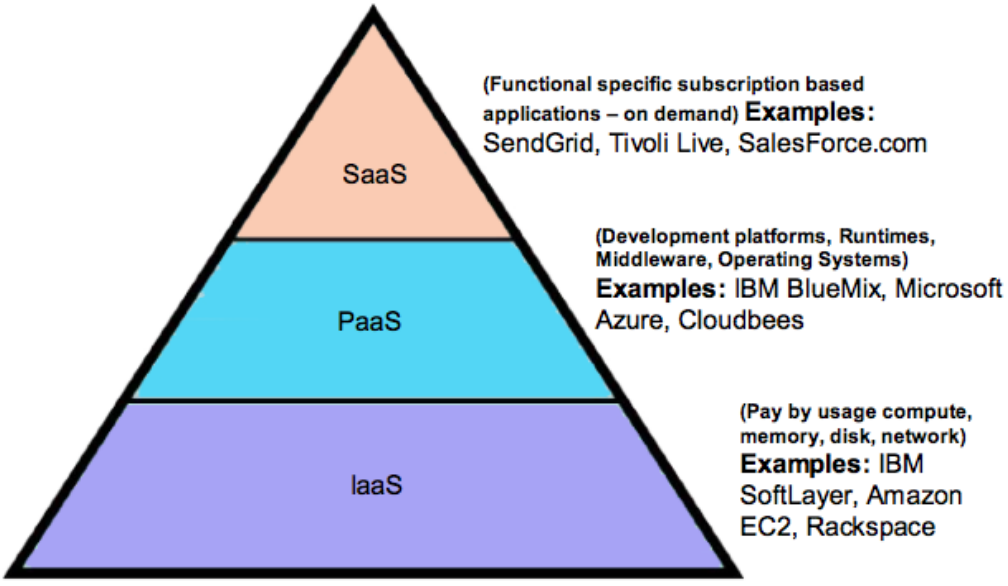


FIGURE 1.1: Cloud Layers[1]

1.2.1 Different Layers of Data Centers

In order to get a better comprehension over the three categories of data center services, an inclusive description of them is provided in the following with examples [7]:

[1]Software as a Service
[2]Platform as a Service
[3]Infrastructure as a Service

1.2.1.1 SaaS

In some ways, SaaS is extremely similar to a lightweight computer built with the purpose of remote controlling the servers. In this case, the clients are normally web browsers and provide access point to software running on servers. SaaS is the most familiar configuration of cloud services for customers. SaaS transfers the function of software administration and its installation to third-party services. Among the most well-known SaaS utilization for commerce are customer relationship management applications. Examples for the application this type of service are comprehensive such as Salesforce, productivity softwares like Google Apps, and storage solutions like Dropbox.

Use of SaaS applications tends to minimize the cost of software right of ownership by eliminating the requirement for technical staff to administer installation, management, and upgrade of software, as well as reducing the cost for licensing. SaaS applications are usually provided on a subscription plan which will be discussed later on in this chapter.

1.2.1.2 PaaS

PaaS operates at a lower tier of the data center than SaaS, typically offering software developers a platform, where they can be develop and deploy their softwares. PaaS providers remove most of the complexities of handling the servers and provide clients with an environment in which the operating system and server software are taken care of. In addition, they handle the low level server hardware and network infrastructure, leaving the customers to focus on alteration of business plan due to scalability, and the application expansion of their own product or service.

As it is the case with majority of cloud services, PaaS is built based on virtualization technology. Businesses can order and posses resources as they require them, scaling as demand expands, rather than spending on unessential and redundant resources.

Heroku, Red Hat's OpenShift, and Google App Engine are examples of PaaS providers.

1.2.1.3 IaaS

Moving down the lower levels of cloud computing structure, we arrive at the root building blocks for cloud services. IaaS provides exceptionally automated and scalable hardware resources, accompanied by cloud storage, processor performance, and network capability. These prospects can be self-provisioned, adjusted, and the service is available on-demand.

IaaS providers deliver cloud servers and their associated resources by an API[4] or dashboard. IaaS providers grant their clients direct access to their servers and storage, the same as they would have access with traditional servers. Nonetheless, the clients have access to a higher level of scalability. IaaS offers the opportunity of outsourcing and building a "virtual machine" in the cloud to its clients, while granting access to many of the same technologies and resource capabilities of a traditional data centers, disregarding the cost of capacity planning or the physical management and maintenance.

IaaS is the most adaptable form of cloud computing which permits automated deployment of servers, processing performance, memory storage, and networking. The clients of IaaS have a complete jurisdiction over their infrastructure rather than customers of PaaS or SaaS services. Development and deployment of PaaS, SaaS, and web-scale applications are the basic employment of IaaS services.

There are many small-scale service provider providing Infrastructure as a Service such as Navisite, exoscale, and Softlayer. However, the three main service providers in this area are Azure, Amazon EC2, and Google Cloud Platform.

[4] Application programming interface

1.2.2 Pricing Plans of IaaS

The focus of this study is on the IaaS that is offered by Amazon. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud, that is why we chose this service provider to examine. Amazon has three dominant pricing plans for its IaaS. To distinguish the differences in these pricing models before we define each of these plans, we require to understand the concept of SLA[5]. SLA is a contract between a service provider and its internal or external customers that documents what services the provider will furnish and defines the performance standards the provider is obligated to meet.

SLAs establish customer expectations with regard to the service provider's performance and quality in a number of ways. Some metrics offered by Amazon that SLAs may specify include:

- Availability/uptime - the proportion of time which services will be available.
- Particular performance benchmarks, where true render execution will be compared for every interval.
- Application response time.
- Notification of network changes in advance that might affect user's performance.
- Help desk response time for a variety of problem classes.
- Usage statistics' delivery.

An SLA specifies availability, performance, memory storage and other parameters for distinct classes of service infrastructure. Now by taking SLA into consideration we can distinguish the pricing plans more easily [8].

[5]Service Level Agreement

1.2.2.1 Subscription

A subscription-based pricing model, also referred to as reserved, is a billing payment plan that allows the user or business organization to purchase (in other words subscribe to) a retailer's cloud services for a designated amount of time for a set price. Subscriptions for a service are usually established on a monthly or annual basis.

This pricing plan for cloud computing is being utilized on a large scale. In a subscription-based scheme, cloud customers typically pay an upfront fee, prior to receiving the requested cloud service. Prices in this plan are often in accordance to the duration of subscription, the longer its term the less its price per time unit will become.

For example, in amazon web services there is an upfront fee required which reserves an instance that you leased and based on its SLA whenever you require the service you reserved the cloud provider is obligated to deliver the requested service to you. It is worth mentioning that aside from the upfront fee, you will be charged with only the billing periods where you have utilized your reserved instance.

1.2.2.2 On-demand

On-demand service procurement, is the standard form of pricing models where the end-user will purchase the instance over a random amount of time for a specific service which they requested. On-demand as defined by the National Institute of Standards and Technology, is the process through which "a consumer (or any user for our purposes here) can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider".

It is due consideration that regarding on-demand SLA the service provider is compelled to allocate the requested capacity to you, during the time you paid the service fee for.

1.2.2.3 Spot Instances

Spot instances enable you to bid on unused capacity, which can lower your costs significantly. The hourly price for a spot instance is set by the service provider, and fluctuates depending on the supply of and demand for spot instances. Your spot instance runs whenever your bid exceeds the current market price.

Spot instances are a cost-effective choice if you can be flexible about when your applications run and if the applications can be interrupted. For example, Spot instances are well-suited for data analysis, batch jobs, background processing, and optional tasks. This is due to their SLA, which indicates that the service provider can kick them out of the system whenever they lack capacity or when their bids fall under the market price.

The key differences between spot and on-demand pricing plans are that spot instances might be dropped out of the system, the term price for spot instances varies based on its demand. In addition, the service provider can terminate an individual spot instance as the hourly price for, or availability of, spot instances changes. One strategy is to launch a core group of On-Demand instances to maintain a minimum level of guaranteed compute resources for your applications, and supplement them with spot instances when the opportunity arises.

1.3 Problem Description

As we discussed before, the application of cloud computing is on the rise nowadays. Thus, we need to consider different aspects of these systems in order to improve them. Many researches have been conducted on every feature of a cloud

system, in two approaches; one, considering the service providers perspective, the other takes the end-users satisfaction in to account. Nonetheless, a great proportion of these studies has been done on the revenue of the cloud provider and the pricing of the services. We can see some of the works that has been done on this subject in a survey that Al-Roomi *et al.* has presented in 2013 [9]. They consider many factors in cloud pricing and service offering. One of the crucial factors that they discuss is called QoS[6], which specifies the technological equipments and services that the provider is offering - the more advanced technology used in a cloud the higher the prices will be. The other factor which has drawn many researchers' attention is the scheduling and allocation of its resources [10, 11]. Nonetheless, few analysis has been done over while excelling revenue by resource allocation.

To this accord, the goal of this study is to investigate how revenue will alter by changing the resource allocation. Furthermore, we will examine the effect of several pricing policies in total revenue of a service provider. In the following we will provide a comprehensive and detailed description of the problem, including the assumptions of the problem.

First of all, we consider the Amazon EC2[7] as our base service provider and we will build the model and the assumptions around it. Amazon EC2 offers several families of IaaSs which they are called differently according to their purposes. For example, they offer a service called **M4** which provides a balance of compute, memory, and network resources, and it is suitable for many applications. The features of this model are given below:

- 2.3 GHz Intel Xeon® E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors
- EBS-optimized by default at no additional cost
- Support for Enhanced Networking
- Balance of compute, memory, and network resources

[6] Quality of Service
[7] Amazon Elastic Compute Cloud

Moreover, we can see the attributes of different instances offered by this family in the following table:

Table 1.1: The attributes of M4 models offered by Amazon

Model	*vCPU*	*Mem (GiB)*	*SSD Storage (GB)*	*Dedicated EBS Bandwidth (Mbps)*
m4.large	2	8	EBS-only	450
m4.xlarge	4	16	EBS-only	750
m4.2xlarge	8	32	EBS-only	1,000
m4.4xlarge	16	64	EBS-only	2,000
m4.10xlarge	40	128	EBS-only	4,000
m4.16xlarge	64	256	EBS-only	10,000

We will assume in our analysis that the provider is offering only one service for simplification purposes. We have chosen `m4.large` as the base service in this study.

Subsequently, after the customer's decision on which service he or she will go for, again the user has three options for pricing plans. Choosing each pricing scheme both relies on the customer's preference and the QoS that he/she requires. Reserved instances are chosen when the user wants to be certain about the availability of the service they need in time. Thus, they will subscribe the service for a certain period of time (we suppose that the subscription's term is fixed and equals 30 days). Obviously this is the most expensive option. If the customer prefers a more cost-efficient service, while their applications are not required on a specific time-line; however, they cannot afford any interruptions, on-demand instances become a more suitable option for the customer. Nevertheless, when the need of the users for cloud services does not have many restrictions, they can use spot instances which is the most cost friendly service. Spot instances are most appealing to customers such as developers, that have batch jobs to run on the cloud. Table 1.2, indicates the prices of `m4.large` offered by Amazon.

The crucial factor is that the cloud provider has a limited capacity and it needs to optimally allocate each of its resources to increase the revenue. Note that we suppose the capacity of the system is an integer and indicates the number of m4.large instances which can be allocated to the system. Moreover, the three

Table 1.2: m4.large Pricing Plans

Pricing Plan	*Upfront Fee (for 30-day subscription)*	*Daily Rate*
Reserved	$45.1	$1.86
On-demand	$0	$3.24
Spot	$0	$0.3144

pricing plans have their own unique demands at each time. In order to make verifiable comparison and analysis, we introduce two definitions on the subject of time-line. First, the term time interval is defined as one day. Second, we set a time horizon as 360 days (in another words, 360 time intervals).

According to the assumptions mentioned in the previous paragraph, we can continue with the problem description. We have referred to resource allocation as one the issues that concerns a service provider. Aside from that, a more complex matter is how to invest the resources to each instance type. This means that besides the fact how many of the instances should be accepted to the system, what is the best policy for accepting each user. This becomes a crucial factor when the service provider is deciding on which instance type is best to be accepted. Furthermore, the policy and the price of spot market is a crucial concern which requires a diligent investigation.

In this study we have utilized simulation techniques to compare and find the best service allocation and pricing plan under several circumstances. Initially, we verify the proposition of Toosi *et al.* in their 2015 paper, indicating the proportion of reserved instances accepted by the service provider will influence the revenue produced for the cloud in the long run. In addition, the contribution of this thesis is that first we remove some of the simplifications assumed in Toosi's work and introduce a more realistic revenue procurement. Moreover, we investigate the impact of several pricing policies for on-demand and spot instances on the total revenue. We have incorporated the concept of threshold for spot instances and analyzed it in an auction based mechanism. For on-demand customers we introduced a queue where in case of capacity shortage will be activated and when

empty capacity becomes available they can enter the system with a discounted price.

The following chapter is dedicated to discussing the previous work that has been done on the subject of data center's revenue improvement and cloud computing on broader sense. In chapter 3, we will present the models, and pricing policies that have been analyzed throughout this thesis. Chapter 4, presents the computational environment, the datasets used for simulation, and illustrates the results and the findings of the models and pricing policies used in this study. Finally, the last chapter includes the discussion on the conclusion and future work which can be conducted on this subject.

Chapter 2

Literature Review

Revenue management is defined as the application of disciplined analytics that predict consumer behavior at a micro-market level and enhance product availability and price to optimize revenue in a system with fixed capacity of perishable resources by market demand management and scheduling. During the last few decades, this subject has witnessed a significant growth both in a scientific and practical sense, especially in the airline and hotel industries [3]. The vital factors that are considered in a revenue management analysis are the demand, selling price, inventory allocation, and market segmentation [3].

The concept of market segmentation was presented to the scientific world in 1956 by WR Smith, as the process of dividing a market of potential customers into groups, or segments, based on different characteristics. The segments created are composed of consumers who will respond similarly to marketing strategies and who share traits such as similar interests, needs, or locations [12]. As the literature on the subject of revenue management is extensive, we focus on its relevant applications to cloud computing. Note that the concept of market segmentation should be well considered in each model that is proposed for a cloud revenue model [13].

Recently, cloud revenue management has attracted many researchers' attention. Both optimal pricing and scheduling of the data centers in order to maximize the revenue have been a trending subject in recent literature. In 2014, Heiling *et*

al. examined the literature to specify the recent work on different aspects of cloud computing in terms of decision analysis [14]. One major factor for decision makers is a pricing scheme for various services of cloud providers. They specify an important characteristic of cloud computing as on-demand access and call it *accessibility*. This of a high importance, has five major considerations:

- A benefit of cloud computing is automated updates from the provider (in other words Version Control) that get pushed down to the customer's systems. While these often bring improvements and security in terms of data loss, they can also disrupt accessibility and can cause difficulties to push the data back to cloud. A rudimentary example for this feature is the businesses based on Dropbox where your files are both on the company's computer and the cloud's servers.

- Because cloud applications have to be accessed through a connection users often have to go through a browser, which adds another layer to the "accessibility value chain", the bandwidth of the user's connection becomes a crucial factor.

- Having an integrated platform is beneficial for an organization, but can cause problems for people who lacks in computer skills, where they have to become experts in that platform and the associated accessibility options.

- Thin clients — in which the device is merely used to display information and receive inputs, while the actual computing takes place on cloud servers — are an essential part of many cloud systems. However, this option can cause the client problems, since it may reduce the accessibility of the cloud.

- Visualizations are a useful tool to display large data sets in a more user friendly fashion for the user. There are tools and methods for getting around this but those are not always available in a cloud setup.

Moreover, Heilig mentions *pricing* as a significant characteristic in cloud (on-demand delivery of cloud services requires a flexible pricing scheme). In addition to

revenue maximization, the model is required to be assessed comparing its SLAs, it can be done also with standardization and Virtualization[1] [15]. It is worth mentioning that in these two studies they have not considered market segmentation and only set up their models on on-demand services. In the following we investigate the most recent literature that have approached the pricing and scheduling of Cloud providers in a dynamic manner.

There has been a vast literature in cloud computing regarding the pricing of cloud services. In 2016, Do *et al.* devised a model for price competition among a heterogeneous cloud market incorporating the multi-tiered interaction between the users and the cloud service providers. They claim that their model examines the interplay of overcrowding, pricing and performance. Do *et al.* maintain that the queuing model that they have used in their paper ($M/M/\infty$ queue) has been adopted in several other papers on data centers. In detail their model extracts the equilibrium prices of the non-cooperative static game and successively the dynamic of the cloud users in the selection game [16].

Sharma *et al.* assess the fair prices regarding both cloud providers and customers, and came up with a pricing architecture which is derived based on financial options theory. The prices are adjusted using financial value-at-risk (VaR) analysis; they used a genetic algorithm to calculate the value-at-risk and incorporated it in their pricing scheme [17].

In 2010, Shang *et al.* saw the rising of cloud computing services, and came up with auction based pricing policy. They address the flexibility of this market well; however, since the market's size was small at the time they sufficed to a static model. They used a double auction Bayesian game based algorithm to obtain the optimal pricing policy. One major restriction in their model is that they consider the price of the providers and the users are uniformly distributed [18].

The transition from static pricing schemes to dynamic pricing schemes in cloud computing services was observed by Li in 2013. These pricing techniques are based on the available resources and the market. They evaluate the performance

[1]Virtualization lets you ***run*** multiple operating systems and applications on a single server

of several allocation algorithms for virtual machines (they call the instances that the service provider offer as virtual machines[2]) considering that the providers has adopted a dynamic mechanism for pricing policy [19].

The main effort on dynamic pricing of cloud services has been on spot instances, e.g. Amazon EC2 services which the prices are dynamically changed. Xu and Li published a paper on dynamic pricing schemes for spot instances. They used dynamic programming techniques to maximize the revenue of the service provider. Their model eliminates the nature of auction in spot instances market, which subsequently will remove the possibility of removing the user from the system[4]. Toosi *et al.* published a paper in 2015, implementing the Ex-CORE auction algorithm and allocating the virtual machines for a spot instance provider as an online software called SipaaS[3]. The proposed framework is tested upon OpenStack dashboard, and evaluated [20].

In 2015, Alzhouri *et al.* presented a dynamic programming algorithm to estimate the pricing of IaaS of a cloud provider. They define their model as Markov decision process under uncertainty on a finite time horizon. In order to overcome complexity of the problem they discretized the time horizon. They applied their formulation on the spot instances market to maximize the profit. However, regarding the uncertainty of the arrival and departure of the demands lead to a complex formulation which cannot be calculated in real time [21].

Similarly, Jin *et al.* introduce two major issues in current cloud computing pricing scheme (i.e., pay-as-you-go for spot instances) in IaaS platform and try to ameliorate these issues by their proposed model; (i) the profit of cloud provider and the user usually are in contradiction with each other; (ii) another issue is the VM's[4] maintenance overhead cost such as start up cost are neglected. This model determines a price that is both satisfactory to both the CSP and the customer [22]. Nevertheless, they do not consider all three cloud services[5] in the revenue maximization models.

[2]VM
[3]Spot instance pricing as a Service
[4]Virtual Machine
[5]Reservation, on-demand, and spot instances

In addition to dynamic pricing and revenue maximization, there are some papers and studies that consider resource allocation and scheduling. Lin *et al.* present a dynamic auction mechanism targets the allocation problem of spot instances and their bids. The optimal method to accept the bids which will result in the highest revenue possible [23]. They defined a threshold in a system with n customers and a cloud service provider, and each user makes a bid. The CSP[6] will accept the these bids based on the capacity available and those that are accepted will be allocated a VM. The threshold will be a value higher less than the minimum value of bids that has been accepted, and higher than the maximum value of the bids that has been rejected.d

In addition, the deficiency of incorporating the cloud capacity scheduling for revenue optimization purposes is a none refutable fact in most literature. In this regard, Toosi *et al.* address the capacity allocation of a data center's resources on a finite horizon, where they offer three plans of on-demand, reservation (i.e., subscription) and spot market. They try to maximize the revenue by solving an optimization problem, in which they employ stochastic dynamic programming, in order to optimize the capacity allocation of each pricing plans. Also, they used heuristic algorithms to optimize computational complexity. The model was adapted from a large-scale real-life problem of Google; on the other hand, they use heuristics to allow online decision making take place in real time [24]. However, Their model lacks authenticity due to its oversimplification estimating the reservation utilization and the pricing fluctuation of spot instances.

We have based our simulation analysis on the model that is proposed by Toosi *et al.* in 2015, and changed their assumptions to more realistic ones. furthermore, we have employed the models and assumptions used in other works to enhance our simulation analysis. In this manner we need to scrutinize the model used by Toosi *et al.* for a better understanding of the following chapters. They consider the IaaS of Amazon EC2 and they use a heuristic algorithm to find the optimal allocation over the three pricing plans (subscription, on-demand, and spot instances). They study the system over a finite horizon, and divide it to several equally distributed

[6]Cloud Service Provider

time intervals. Next, they assume that the demands of each instance is known prior to their analysis. And move on to, defining a decision variable on what proportion of the reserved instances they accept to the system. Their algorithm will receive reserved instances as they are the most guaranteed form of revenue [24]. In the next iteration they will accept as many on-demand instances as possible, this due to the fact that on-demand instances produces almost the same as reserved consumers. This assumption is easily authenticated by calculating the their revenue over 30 days. We use the prices offered by Amazon EC2 in table 1.2. The revenue of a reserved instance in this case will be \$100.9 and it will be \$97.2 for on-demand instances; however, this value for a spot customer is \$9.432. They assume that the spot users will only receive their service for one time interval which is an over simplification in their study. Furthermore, they assume that the rejected customers are lost forever.

Based on the previous studies that have been mention in this chapter, we have proposed a simulation analysis of the cloud service provider's revenue. This study will analyze different policies that a CSP can take, in terms of the its revenue. In the following chapter we will present our simulation model and its assumptions.

Chapter 3

Simulation Model

This chapter includes the assumptions and the logic that our model is based on. Then, we discuss the complete models, and the results and findings of the simulation model in the next chapter.

3.1 System Model

In this section, we discuss the different types of the services that are offered by a typical cloud service provider such as Amazon EC2 Web Services. We assume that the service provider has only one type of instance (`m4.large` of Amazon EC2). The prices used in this model are derived from table 1.2.

3.1.1 Cloud Pricing Plans

The model at hand considers three pricing plans which their description is provided in the following:

3.1.1.1 On-Demand Instances

These are also referred to as pay-as-you-go plan are vastly used by cellphone operators. With on-demand instances, you pay for computing capacity by the day with no long-term commitments or upfront payments. You can increase or decrease your computing capacity depending on the demands of your application and only pay the specified daily rate for the instances you use. The user is charged at a fixed rate ($p = \$3.24$) per billing cycle. The rate of on-demand instances is stable and does not change for most IaaS providers [24].

3.1.1.2 Reserved Instances

This plan requires an upfront fee ($\varphi = \$45.1$), and will guarantee a space on the server for a specific duration which is stated in the reservation contract. The time span of these plans are typically one month (we assume them to be 30 days). The plan is said to be live when the user is utilizing its capacity. In that case, the provider charges the user a discounted daily fee compared with the daily fee of the pay-as-you-go plan. The amount of user's usage of its capacity is accumulated and used to calculate the bill. The discount factor is manifested as $\alpha = 0.574$ (the value of α is the ratio of reserved daily rate over on-demand daily rate) and the daily usage is shown by t, thus the bill of a reservation plan user will be $\varphi + \alpha\, p\, t$.

3.1.1.3 Spot Instances

In this plan, the customers will bid on instances in an auction based market. If their bid was higher than the threshold of the IaaS provider, then they are granted the access to the instance. However, this plan has the lowest QoS[1]. This means that, in case of capacity shortage or overload in the system, they can be dumped out of the system. Nonetheless, this plan has attracted many customers due to the fact that the user can save up to 90% in comparison to on-demand instances [25]. Spot instance's price is regulated by a discounted rate $p_s = \$0.3144$, and the

[1] Quality of Service

price at each billing cycle is quantified by their bids. Using these pricing plans and their we can continue to the base model that we used in the simulation.

3.2 Proposed Model

The model that we have conceived simulates the cloud system and examines the dynamics of the system and the revenue of the provider for different policies. Such a simulation will allow us to evaluate each pricing policy and the relationship of price with available capacity. For the sake of clarity, we suppose that the IaaS provider only offers one package (`m4.large` instance (Linux, US east), offered by Amazon Web Services).

Initially, we presume that the planning horizon is finite and decomposed into τ identically sized time intervals that are equal to the billing cycles of the cloud provider (one day). Time at the beginning of the horizon is set as 0 ($t = 0$) and the demand at each time interval for the three different pricing plans are; $d_0^o \dots d_{\tau-1}^o$ for on-demand instances, $d_0^r \dots d_{\tau-1}^r$ for reserved instances, and $d_0^s \dots d_{\tau-1}^s$ for spot instances. We assume that the demands of each period arrive at the beginning of that time interval. Due to lack of access to demand data, we had to generate the demand for the three services; reserved instances, on-demand instances, and spot instances.

We generated the demands according to a logical pattern, the demand will decrease on weekends and New Year's holidays. We also consider the case where the arrival rates are without fluctuation. The distribution to which the demands are produced with Poisson distribution, this distribution is widely used for arrival rate customers [26]. In the first case, the parameters used to produce the demands are divided in 3 categories. The first category is indicating the demand at a regular rate; this includes the all the periods except at weekends at New Year's holidays (μ_r^s, μ_o^s, and μ_s^s). The second category demonstrates the time when demand is at its lowest rate, this happens on the weekends (μ_r^w, μ_o^w, and μ_s^w). The last category is showing the demand on holidays when there is a decline in the rate of

arriving customers (μ_r^h, μ_o^h, and μ_s^h). The other case is when the demands are at a constant arrival rate (μ_r, μ_o, and μ_s). Note that the customers will all arrive at the beginning of the time interval into the system.

The goal of the cloud service provider is to optimally allocate its capacity to ensure the highest possible revenue. In this section, we present the mathematical formulation that supports our simulation model. We assume the capacity of the system to be C which means the servers in the system can contain up to C instances.

The cloud service provider decides on what percentage of the reserved instances it accepts at each time period (ρ). The reservation plan is the most stable line of income for the cloud provider, also the most profitable per each customer. We assume that any accepted reserved instance will be stay in the system for 30 days. So in order to figure out how much capacity we have for the other instances, we need to know how many reserved instances we accept in the last 30 days. An accepted reserved instance will not always actively use the capacity it reserved and for each time period it will utilize the computing resources with a probability of normal distribution with parameters μ & σ. Normal distribution is chosen for its bell-shaped symmetric characteristic. We generate the utilization alongside the demands before running the simulation, in order to compare different policies objectively.

The capacity remaining once the reserve instances are accepted, can be used for on-demand and spot instances. The life time of an on-demand instance is assumed to have a geometric distribution with the average of 11 days. Since they are more profitable compared to spot instances, depending on the remaining capacity and arriving demand of on-demand instances, we accept as many on-demand instances as possible. Whereas reserved instances may some times not fully utilize the allocated capacity, on-demand instances will fully employ the computer resource they have requested.

Finally, based on the remaining capacity and demand of spot instances, we accept as many spot instances as possible to maximize capacity utilization. Spot

instances due to its SLA can be kicked out of the system at each time when there is an overload in the system, thus they can utilize the capacity that is reserved while there is no active instance. The spot instances' life time are estimated with a geometric distribution. The average time that spot users stay in the system is 5.5 days. We designed the system in a manner that in case of an overload at an interval it will exterminate spot instances equal to the number of overload instances.

Our goal is to find the best ρ under different policies and circumstances. We define r_t as the number of demands that are accepted to the system at time t. The simulation model will run for different proportions of accepted reserved instances to see which policy is the most profitable with regards to the cloud provider.

The lifespan of the reserved instances is less than the time horizon of the system. We suppose that the time horizon of the system equals 360 days, and the lifespan of reserved instances is 30 days (ν). Due to the this argument, the system is required to update itself after the life time of these instance finishes. Note that the upfront fee of reserved instances in the last month should not be fully accounted for in the revenue calculations of the year. To resolve this issue we multiplied the upfront fee with a linearly decreasing coefficient in the last month (ψ) that will only consider the share of the upfront fee in regards to the hours left into the time horizon.

3.2.1 General Assumptions

According to the theoretical model explain prior we can present the general assumptions used in this study in the list below:

- We run each scenario on three different capacities; first, the case where we have capacity shortage (1500). Second, we simulated on capacity higher than the average resources that customers occupy (2500). Finally, we evaluate the revenues on excessive scarcity of resources (1000). The average capacity

is calculated by the summation of three arguments; (i) multiplication of reserved average demand by 30 day of subscription period and the mean utilization overtime, (ii) average demand of on-demand instances per period times their mean service duration, (iii) the product of average spot demand and their service duration.

$$\textit{maximum average capacity} = m_u\,(30\,\mu_r) + 11\,\mu_o + 5.5\,\mu_s$$

- We assume that the rejected customers or the ones that leave the system will not reenter the cloud. This assumption is changed in some of the cases studied, which is explained later.

- The decision variable of the model is the percentage of accepted reserved instances at each interval. We try different values of ρ and pick the best rho for each policy, demand data and capacity combination (ρ = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)

- At time 0, we have $l_0^r = 0$, $l_0^o = 0$ and $l_0^s = 0$. And, the demand starts arriving at the first period.

- We assume that each on-demand instance will stay in the system according to a geometric distribution with parameter 0.09 which means on the average they will leave after $\mu_\omega = 11$ periods.

- Moreover, the lifespan of spot instances is generated by geometric distribution with average of 5.5 days (μ_ε).

- As the subscription of reserved instances is for 30 days, we define a discount factor ψ_t for the last 29 days. Thus, we incorporate only the proportion of the upfront fee which is in our time horizon.

- Note that the instances arrive at the beginning of each time period and they will either get accepted or rejected by the system. Those customers that were disapproved will disappear from the system and will find another service provider.

- Reserved instances may utilize their secured capacity. Otherwise, this capacity can be allocated to spot instances (Due to their SLA) when there is a shortage. It is worth mentioning, we cannot allocate any on-demand instances to this proportion of the system, since according to SLA of on-demand instances they cannot be removed from the system. Elimination of the customers allocated to the unused part of the capacity is the result of the increase in utilization of reserved instances or accepting new instances to the system in an upcoming period.

3.2.2 Model Description

Moving on to the main models we used for the simulation analysis, we have devised 8 models to test their revenue over the time horizon of 360 days. These algorithms are based on the method used by Toosi *et al.* in 2015. We start by Toosi's model and move on to more detailed and realistic model that we proposed in this thesis.

3.2.2.1 Model 1.0

This is the exact model that Toosi *et al.* proposed in their 2015 paper. We set this model as a base structure for our proposed models. Assuming the demands of each instance for all time intervals, and the utilization of reserved instances for each time interval is known prior to the execution the model, we continue to the first iteration. Note that we will calculate revenue for all values of ρ which are stated in general assumptions.

First step of the model is to find the number of reserved instances that can be received to the system. The number of reserved instances accepted to the cloud at time t[2] is assigned to the variable r_t, and we show the number of reserved instances that are currently in the system by l_t^r). Similarly, The number of on-demand instances in the clouds from previous time intervals is manifested by l_t^o.

[2]'t' subscript indicates the time interval

So, the value of the r_t can be computed by equation 3.1. This formulation is indicating that the cloud can only accept the minimum of ρ percentage of the new customers or in case of capacity shortage it will receive a number equal to its left capacity.

$$r_t = min(C - l^r_{t-1} - l^o_{t-1},\ \rho\, d^r_t), \qquad \forall t \in (1,\, \tau) \tag{3.1}$$

It is worth mentioning that the value of the term on left side of this equation never gets negative values, since it did not accept more than its capacity in the prior period. l^r_t requires to be updated after each iteration according to:

$$l^r_t = l^r_{t-1} + r_t, \qquad \forall t \in (1,\, \nu) \tag{3.2a}$$

$$l^r_t = l^r_{t-1} + r_t - r_{t-\nu}, \qquad \forall t \in (\nu,\, \tau) \tag{3.2b}$$

This is based on the fact that the instances will leave the system after they finish their 30 day (ν specifies a 30 days or in other words 30 intervals) subscription. In the next iteration we will compute the value of o_t which indicates the number of on-demand instances that the model will introduce to the system.

$$o_t = min(C - r_t - l^r_{t-1} - l^o_{t-1},\ d^o_t), \qquad \forall t \in (1, \tau) \tag{3.3}$$

On-demand instances have a limited duration individually (ω_j) with a geometric distribution. For each on-demand instance j, we define a variable $e_j = t_j + \omega_j$, where t_j is the period in which the instance is accepted to the system and e_j is the last period before it leaves the system. The model requires to update the number of live on-demand instances after each one is terminated. The number of the terminated instances at time t are subtracted from the number of live on-demand instances, i.e. $l^o_t = l^o_{t-1} + o_t - \kappa_t$, where κ_t is the number of on-demand instances that leave the system at time t and is calculated as $\sum_j \chi_j(t)$ with $\chi_j(t) = 1$ if

$t = e_j$, 0 otherwise.

$$\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j \\ 0 & \text{otherwise} \end{cases} \tag{3.4a}$$

$$\kappa_t = \sum_j \chi_j(t) \qquad \forall t \in (1, \tau) \tag{3.4b}$$

$$l_t^o = l_{t-1}^o + o_t - \kappa_t \qquad \forall t \in (1, \tau) \tag{3.4c}$$

Following the iterations, s_t will be specified which is the number of accepted spot instances. It is done by the same logic as for reserved and on-demand customers. Moreover, the decision of accepting spot instances rely on the utilization of reserved users, which those are employing their capacity are called live reserved instances. The utilization of these customers are designated beforehand while generating the demands, for the sake of comparison of each model. The utilization at each time interval is assigned to the variable u_t The utilization at each time interval is generated by a normal distribution with parameters μ_u and σ_u, and only those that will stay in the interval of $[0, 1]$ will be accepted by the model. The parameters of this normal random generator are chosen in fashion, so the values generated do not exceed the limit stated previously.

Toosi has proposed that the spot instances will be live in the system for only one period, thus there will be no l_t^s [24]. l_t^s indicates the number of spot instances at time t that are live in the system, which has a role in the upcoming models.

$$s_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o,\ d_t^s), \qquad \forall t \in (1, \tau) \tag{3.5}$$

At this point, the model computes the total revenue of the system at each time interval over the horizon (360 days) for a specific ρ. Equation 3.6a calculates the revenue generated at each iteration. However, this is the income for the billing hours throughout the 360 days, except for the last 30 days (30 days is shown by ν in the formulation) where we use the ψ coefficient to adjust the revenue in the 30 days (see equation 3.6b). This is regarding to the fact that we intend to calculate

the revenue over the time horizon and only the proportion of upfront fee should be considered in the revenue which resides in the time horizon.

$$\zeta_t = \varphi r_t + \alpha p u_t (r_t + l_t^r) + p(o_t + l_t^o) + p_s\, s_t, \qquad \forall t \in (1, \tau - \nu) \tag{3.6a}$$

$$\zeta_t = \psi(t) \varphi r_t + \alpha p u_t (r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + p_s\, s_t, \qquad \forall t \in (\tau - \nu + 1, \tau) \tag{3.6b}$$

One of the simplifications in Toosi's model is that they assume the price of spot instances is a constant. p_s is the price of spot instances. However, the price of this service is different for each user and equals their bid, which we will explain in details in the following models. By having the value of revenue for each time period we can find the revenue of the system on the complete time horizon.

$$Z = \sum_{t=1}^{\tau} \zeta_t, \qquad \forall t \in (1, \tau) \tag{3.7}$$

Algorithm 1 illustrates the pseudo-code of this model. Moreover, in appendix A we have assembled a list of all the variables and parameters used throughout this study with their descriptions, which includes the parameters used in the pseudo-codes.

3.2.2.2 Model 2.0

This model is based on the one realistic assumption which is the only factor that differs from the previous version. In this model we assume that spot instances can stay in the system for several periods and these instances are chosen in an auction based mechanism by the cloud service provider. In other words, the spot instances are chosen based on the bids that they make on this service. In order for spot customers to make their bids, the cloud provider should provide them with a threshold which is the derived from historical data. We assume that at every iteration the service provider will announce the threshold for the last

Algorithm 1 Model 1.0

procedure REVENUE CALCULATION BASED ON ρ AND CAPACITY:
  Inputs $\leftarrow \vec{d_r},\ \vec{d_o},\ \vec{d_s}$, and $\vec{u}$
  Prices $\leftarrow p$, α, and p_s
  Initial Parameters $\leftarrow l_0^r = 0,\ l_0^o = 0$
loop over the horizon:
computing the decision variables:
  $r_t = min(C - l_{t-1}^r - l_{t-1}^o,\ \rho\, d_t^r)$
  $o_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o,\ d_t^o)$
  $s_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o,\ d_t^s)$
computing the revenue of at each time interval:
  if $t \leq (\tau - \nu)$ **then**
    $\zeta_t = \varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + p_s\, s_t$
  else
    $\psi(t) = \frac{1}{\tau}(t - (\tau - \nu))$
    $\zeta_t = \psi(t)\varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + p_s\, s_t$
update the number of live instances in the system:
  if $t \leq (\tau - \nu)$ **then**
    $l_t^r = l_{t-1}^r + r_t)$
  else
    $l_t^r = l_{t-1}^r + r_t - r_{t-\nu}$
  $\omega_j = random.geometric(parameter = \frac{1}{11})$
  $e_j = t_j + \omega_j$
  $\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j 0 \quad \text{otherwise} \end{cases}$
  $\kappa_t = \sum_j \chi_j(t)$
  $l_t^o = l_{t-1}^o + o_t - \kappa_t$
compute the total revenue:
  $Z = \sum_{t=1}^{\tau} \zeta_t$

period. The value of the threshold stated by the cloud is a constant in this model. For computational purposes, we have chosen the value provided by Amazon Web Services in table 1.2. The threshold is represented by the variable Δ, and the customers make their bids based on this parameter. Δ illustrates the value of threshold at the previous period supposedly; however, it is a constant value in this case. Moving on to the life span of these individuals, it is generated by a geometric distribution with parameter 1 over the average life span of a single spot instance. This is the base difference between model versions *1.0* and *2.0*.

The procedure of this model is relevantly similar to the previous version. However, as the spot instances can stay in the cloud for their requested period we

will add a parameter l_t^s. This parameter indicates the number spot customers that are live in the system. Initially, the model will designate the decision variables r_t and $\emptyset_t$ the same as model 1.0 by equations 3.1 and 3.3. The simulation algorithm does not consider the number of spot instances in the system, since they can be dropped out of the cloud due to their SLA. Moving forward, the algorithm will generate a bid for all the demands of spot instances at time t (t $\in (1, \tau)$). The generation of each bid is done by Gamma distribution. We have used Gamma distribution due to its property of being a non-negative random generator [27]. This distribution has been used for the generation of bids by a random variable in literature, such as the study by Selçuk & Özlük in 2012 [28]. The average and standard deviation of these bids are μ_π and σ_π. Note that the customer bids on the average higher than the threshold (Δ). Thus, in terms of mathematical formulation it can be represented as $\mu_\pi = \Delta + \epsilon$. The vector $\vec{\pi_t}$ all the bids of the customer at time t, and it is derived as the following:

$$\pi_t^i = random.gamma(\mu_\pi, \sigma_\pi), \qquad \forall i \in (1, d_t^s) \tag{3.8a}$$

$$\vec{\pi_t} = \{\pi_t^i | \quad \forall i \in (1, d_t^s)\} \tag{3.8b}$$

As each customer has a unique bid, the simulation model will treat them as individuals. For every element in $\vec{\pi_t}$ we generate a service duration based on geometric distribution with the average of 5.5 days. The service duration and spot bids are stored in a two dimensional vector:

$$\Pi_t = \{(\pi_t^i, \eta^i) | \quad \forall i \in (1, d_t^s)\} \tag{3.9}$$

Then, the algorithm will check the empty capacity left in the cloud regardless the live spot instances in the system, for potential users. The service provider do not consider the live spot instances, since when there is a capacity shortage based on spot service's SLA, it can drop out the current customers with bids less than

that of the new customers.

$$\varpi_t = min(C - u_t(r_t + l^r_{t-1}) - l^o_{t-1},\ d^s_t), \qquad \forall t \in (1, \tau) \tag{3.10}$$

The elements of Π_t are stored in a pool (ϑ_t), which is empty at the beginning of the horizon. Next, the cloud will choose the ϖ_t highest bids of the pool (ϖ_t indicates a number). First, we sort the elements in ϑ_t based on the bids in a decreasing manner, and keep the number of elements equal to ϖ_t and eliminate the rest of them. The size of this two dimensional vector indicates the value of l^s_t.

The rest of this model is mainly the same with the previous model, except for updating the spot instances when they leave the system and calculation the revenue for each interval. At the end of each iteration, the model will update the remaining service time for each spot user:

$$\vartheta_t(m, 2) = \vartheta_t(m, 2) - 1, \qquad \forall t \in (1, \tau) \tag{3.11}$$

If the remaining service duration becomes 0 then it will be eliminated from the cloud. As for the revenue of spot instances, it can be computed by summation over the bids in ϑ_t. So, the formulation of revenue of spot instances will become:

$$\zeta^s_t = \sum_m \vartheta_t(m, 1), \qquad \forall t \in (1, \tau) \tag{3.12}$$

By having ζ^s_t the total revenue at each period is computed and stored in ζ_t (the revenues of reserved and on-demand instances are calculated similarly to the previous model). Finally, the total revenue is derived by equation 3.7 as in the previous model.

3.2.2.3 Model 2.1

This model is devised for the circumstances where there is capacity shortage in general. Also, the demands fluctuate over time, resulting in empty capacity at

Algorithm 2 Model 2.0

procedure REVENUE CALCULATION BASED ON ρ AND CAPACITY:
  Inputs $\leftarrow \vec{d_r},\ \vec{d_o},\ \vec{d_s},$ and $\vec{u}$
  Prices and threshold $\leftarrow p$, αp, and Δ
  Initial Parameters $\leftarrow l_0^r = 0$, $l_0^o = 0$, and $\vartheta_0 = \emptyset$
loop over the horizon:
computing the decision variables for reserved and on-demand instances:
  $r_t = min(C - l_{t-1}^r - l_{t-1}^o,\ \rho\, d_t^r)$
  $o_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o,\ d_t^o)$
computing the bids vector and their service time:
  $\mu_\pi = \Delta + \epsilon$
  $\vec{\pi_t} = \{\pi_t^i = random.gamma(\mu_\pi, \sigma_\pi) | \quad \forall i \in (1, d_t^s)\}$
  $\Pi_t = \emptyset$
  for i in Dimension(π_t) **do**
    $\Pi_t = \Pi_t + (\pi_t(i), random.geometric(\frac{1}{5.5}))$
  $\vartheta_t = \vartheta_t + \Pi_t$
  sort ϑ_t descending based on bids
  $\varpi_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o,\ d_t^s)$
  choose the first ϖ_t elements in ϑ_t, delete the rest
computing the revenue of at each time interval:
  if $t \leq (\tau - \nu)$ **then**
    $\zeta_t = \varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + \sum_m \vartheta_t(m, 1)$
  else
    $\psi(t) = \frac{1}{\tau}(t - (\tau - \nu))$
    $\zeta_t = \psi(t)\varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + \sum_m \vartheta_t(m, 1)$
update the number of live instances in the system:
  if $t \leq (\tau - \nu)$ **then**
    $l_t^r = l_{t-1}^r + r_t)$
  else
    $l_t^r = l_{t-1}^r + r_t - r_{t-\nu}$
  $\omega_j = random.geometric(parameter = \frac{1}{11})$
  $e_j = t_j + \omega_j$
  $\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j \\ 0 & \text{otherwise} \end{cases}$
  $\kappa_t = \sum_j \chi_j(t)$
  $l_t^o = l_{t-1}^o + o_t - \kappa_t$
  for i in dimension(ϑ_t) **do**
    $\vartheta_t(i, 2) = \vartheta_t(i, 2) - 1$
    if $\vartheta_t(i, 2) = 0$ **then**
      eliminate $\vartheta_t(i)$
compute the total revenue:
  $Z = \sum_{t=1}^{\tau} \zeta_t$

some intervals. The cloud service provider offers an opportunity for rejected on-demand instances to stay in a queue for a duration. If there will be empty capacity in the following periods, the customer which has waited in the queue will receive the same service with a discount factor ($\beta = 0.8$) for the price of on-demand. The rest of this model is precisely the same as model 2.0.

We introduce a new vector $\vec{Q_t}$, where t indicates the time interval of the simulation model. This vector has 14 elements, as we assumed that every on-demand customer which will enter the queue will wait for 14 periods and if it is not serviced by then it will leave the system. The on-demand customers arrive to the cloud at the beginning of the period for each time interval. If there is not enough capacity for all of them in the cloud, each rejected customer might stay in the queue with a uniform chance between [0.3, 0.7]. The number of these customers will be added to the $\vec{Q_t}$, e.g. $\{4, 6, 3, 8, \ldots, 7\}$ where 4 indicates the number of customers that have waited for 1 period. And, 7 customers have waited for 14 periods in the queue, if these are not serviced at this period they will leave the system. As these user will receive a discount factor for the price of on-demand instances they are represented by another class of variables. q_t designates the number of on-demand instances accepted at the cloud from the queue, and l_t^q is the number of on-demand instances that are live and were initially in the queue. We assume that the service provider will receive the customers that have waited the longest in the system. The calculation of q_t is as follows:

$$q_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q, \sum_j Q_t), \qquad \forall t \in (1, \tau) \tag{3.13}$$

After this finding the value of q_t, the vector $\vec{Q_t}$ needs to be updated and the customers in the queue which were served by the cloud will be subtracted from it. The service duration of these instances will follow the same procedure as for

regular on-demand instances, thus l_t^q is updated by:

$$\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j \\ 0 & \text{otherwise} \end{cases} \tag{3.14a}$$

$$\kappa_t = \sum_j \chi_j(t) \qquad \forall t \in (1, \tau) \tag{3.14b}$$

$$l_t^q = l_{t-1}^q + q_t - \kappa_t \qquad \forall t \in (1, \tau) \tag{3.14c}$$

By having these variables introduced to our system, equations 3.1, 3.3, and 3.10 will be updated respectively to equations 3.15, 3.16, and 3.17:

$$r_t = min(C - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ \rho\, d_t^r), \qquad \forall t \in (1,\ \tau) \tag{3.15}$$

$$o_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ d_t^o), \qquad \forall t \in (1, \tau) \tag{3.16}$$

$$\varpi_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o - l_{t-1}^q,\ d_t^s), \qquad \forall t \in (1, \tau) \tag{3.17}$$

This model will also affect the formulation for revenue calculation at each period, since there is a new source of income.

$$\begin{aligned} \zeta_t = \ & \varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) \\ & \beta p(q_t + l_{t-1}^q) + \sum_m \vartheta_t(m, 1), \qquad \forall t \in (1, \tau - \nu) \end{aligned} \tag{3.18a}$$

$$\begin{aligned} \zeta_t = \ & \psi(t)\varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) \\ & \beta p(q_t + l_{t-1}^q) + \sum_m \vartheta_t(m, 1), \qquad \forall t \in (\tau - \nu + 1, \tau) \end{aligned} \tag{3.18b}$$

To observe the complete procedure of this model, we have written a detailed pseudo-code of it, which is provided in the following page.

3.2.2.4 Model 3.0 and 3.1

Model 3.0 and 3.1 are the exact replicas of model 2.0 and 2.1 respectively, with a slight difference in the concept of threshold. The threshold that was announced to the customers in the previous models were a constant. However, in reality

Algorithm 3 Model 2.1

1: **procedure** Revenue Calculation Based on ρ and Capacity:
2: $Inputs \leftarrow \vec{d}_r,\ \vec{d}_o,\ \vec{d}_s$, and $\vec{u}$
3: $Prices\ and\ threshold \leftarrow p$, αp, and Δ
4: $Initial\ Parameters \leftarrow l_0^r = 0$, $l_0^o = 0$, $\vartheta_0 = \emptyset$, and $Q_t = \vec{0}$
5: ***loop over the horizon:***
6: *computing the decision variables for reserved and on-demand instances:*
7: $r_t = min(C - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ \rho\, d_t^r)$
8: $o_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ d_t^o)$
9: **if** $o_t < d_t^o$ **then**
10: $Q_t(1) = (d_t^o - o_t) * random.uniform(0.3, 0.7)$
11: **if** $C - r_t - l_{t-1}^r - l_{t-1}^o > 0$ & $\sum_j Q_t \neq 0$ **then**
12: $q_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ \sum_j Q_t)$
13: *computing the bids vector and their service time:*
14: $\mu_\pi = \Delta + \epsilon$
15: $\vec{\pi_t} = \{\pi_t^i = random.gamma(\mu_\pi, \sigma_\pi) |\quad \forall i \in (1, d_t^s)\}$
16: $\Pi_t = \emptyset$
17: **for** i in Dimension(π_t) **do**
18: $\Pi_t = \Pi_t + (\pi_t(i),\ random.geometric(\frac{1}{5.5}))$
19: $\vartheta_t = \vartheta_t + \Pi_t$
20: sort ϑ_t descending based on bids
21: $\varpi_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o - l_{t-1}^q,\ d_t^s)$
22: choose the first ϖ_t elements in ϑ_t, delete the rest
23: *computing the revenue of at each time interval:*
24: **if** $t \leq (\tau - \nu)$ **then**
25: $\zeta_t = \varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + \beta p(q_t + l_{t-1}^q) + \sum_m \vartheta_t(m, 1))$
26: **else**
27: $\psi(t) = \frac{1}{\tau}(t - (\tau - \nu))$
28: $\zeta_t = \psi(t)\varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + \beta p(q_t + l_{t-1}^q) + \sum_m \vartheta_t(m, 1))$
29: *update the number of live instances in the system:*
30: **if** $t \leq (\tau - \nu)$ **then**
31: $l_t^r = l_{t-1}^r + r_t)$
32: **else**
33: $l_t^r = l_{t-1}^r + r_t - r_{t-\nu}$
34: $\omega_j = random.geometric(parameter = \frac{1}{11})$
35: $e_j = t_j + \omega_j$
36: $\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j \\ 0 & \text{otherwise} \end{cases}$
37: $\kappa_t = \sum_j \chi_j(t)$
38: $l_t^o = l_{t-1}^o + o_t - \kappa_t$
39: calculate l_t^q by the same procedure in lines 35-38
40: **for** i in dimension(ϑ_t) **do**
41: $\vartheta_t(i, 2) = \vartheta_t(i, 2) - 1$
42: **if** $\vartheta_t(i, 2) = 0$ **then**
43: eliminate $\vartheta_t(i)$
44: ***compute the total revenue:***
45: $Z = \sum_{t=1}^{\tau} \zeta_t$

this does not hold true. The service providers display the historical threshold, this makes sense when there is a capacity shortage. In this case, the bids that are accepted to the cloud with a higher probability are the ones with an average greater than the threshold. This will result in a increase in the value of threshold when there is a overload in the cloud. Nevertheless, when there is empty capacity for several intervals, this value will decrease.

We can derive the following logic for the value of threshold (Δ), based on the paper presented by Lin *et al.* in 2010 [23]. They proposed that the threshold should be a value less than the least bid accepted to the system in terms of value, and higher than the greatest bid which was rejected by the system. In this manner, by simplification we will choose the bid with the least value as threshold (Δ). As threshold is changeable at each iteration it is represented by Δ_t in these two models. Initially, the value of threshold (Δ_0) is set to Δ (the value used in the previous models). Intuitively, we have set a lower bound and upper bound for threshold. If it becomes less than a specific value the CSP will start losing revenue. On the other hand, if it reaches a value higher than an upper bound it will lose its appeal for the customers.

The algorithm for model 3.0 is the same as model 2.0 except for line 10, where μ_π is calculated. In this case Δ is replaced by Δ_t. Δ_t will equal the last bid in vector ϑ_t, where it shows the bid with the least value. Then the value of μ_π will equal $\Delta_t + \epsilon$. As for model 3.1, it requires the same alteration from model 2.1, and the calculation of $\mu_p i$ happens in line 14 of algorithm 3.

3.2.2.5 Model 4.0, 4.1, 4.2, and 4.3

The spot instances are selected in an auction based mechanism, the criteria for their selection is the bids. However, in these models we introduce a new criteria for choosing the customers. We argue that instead of selecting spot instances based on their bids, we should calculate their expected revenues and choose based on this criteria. The models 4.0, 4.1, 4.2, and 4.3 are copies of models 2.0, 2.1, 3.0, and 3.1 respectively, while incorporating this criteria. The logic behind this proposition

is when we have two choices, one a bid of \$0.6 and service duration of 1 day, and the other with \$0.5 bid and 2 days of duration the second one becomes more appealing. By this criteria we will select the second one with a revenue of \$1, while in previous models we would have selected the first one which has a revenue of \$0.6.

The difference between these new models is that the vector ϑ_t will have another dimension indicating the expected revenue. In previous cases ϑ_t had two dimensions, the first one contains the bid of a customer, and the second indicating the service duration of that customer. However, with models 4.($\dot{x}$) the new dimension is the multiplication of the first two which will provide the expected revenue. Furthermore, the criteria for choosing the top ϖ_t customers out of the ϑ_t set will change from bids to expected revenue. The rest of their algorithms are parallel to previous versions. Table 3.1, illustrates all the models and their differences.

Table 3.1: Factors of each model version

Model Version	*1.0*	*2.0*	2.1	3.0	3.1	4.0	4.1	4.2	4.3
Without Queue		x		x		x		x	
With Queue			x		x		x		x
Constant Threshold		x	x			x	x		
Adaptable Threshold				x	x			x	x
Based on Bids		x	x	x	x				
Based on Expected Revenue						x	x	x	x

In the following chapter, we will provide several datasets and implement these algorithms on them. Furthermore, we will evaluate each model under different circumstances, and discuss the results and findings.

Chapter 4

Datasets and Results

In this chapter we will introduce the corresponding computational environment for the simulations, and then move on to discussing the datasets used for this study and their features. Moreover, we will present the results of each simulation runs and analyze the total revenue produced by each model for every dataset. Finally, we argue about the findings and interpret the results.

4.1 Corresponding Computational Environment

The simulation models were executed on a host computer with a Debian-based Linux operating platform (Linux 16.04 LTS). The configuration of the host is listed below:

- System Name `DESKTOP-VUNIM81`
- System Manufacturer `LENOVO`
- System Type `x64-based PC`
- Processors `Dual-Core Intel(R) Core(TM) i5-5200U Processor`
- Processor's Cache `3 MB`

- Physical Memory `3.74 GB`
- Virtual Memory `7.70 GB`

The models presented in the previous chapter were coded in python programming language and we used python 3.5 as its interpreter. As an example the code of model 3.1 is provided in Appendix B. The codes used for this simulation analysis are accessible to public[1]. In the following we will introduce the datasets used in this study, and discuss their features.

4.2 Datasets

As revenue management is a competitive subject amongst CSPs they do not disclose their demands to public. In this regard, we were not able to receive the demands of service providers. To overcome this, we have provided several datasets where each considers a specific case for the demand. Demands are generated using a Poisson random generator. As for the prices we use the rates provided by Amazon for calculating the revenue (see table 1.2). We ran the simulations on two different capacities 1500, and 2500 to investigate the effect of capacity on determination of optimal pricing policy. The datasets used in this study are presented in the following:

4.2.1 Dataset 1

A CSP will provide three types of instances - reserved, on-demand, and spot - where their demands has an average for each period. In our models we assume that the service provider knows the utilization of reserved instances for each period prior to simulation for a fair comparison. Dataset 1 represents a case where reserved instances occupy most of the cloud's capacity on average regardless of ρ. We hypothesize that each demand average will fluctuate overtime based on the time

[1] https://bitbucket.org/cloudrevenuemanagement/python-codes/overview

of the customers arrival on the horizon. Note that the demands will arrive at the beginning of each period. The arrival rates are divided into three categories. Regular, which indicates the highest rate at normal days. Holidays, specifying a reduced rate for the arrivals in contrast to standard time, this is true for New Year's holidays (14 days) and weekends, when the arrival rates will reduce to the lowest rate. For the sake of comparison, we generate the utilization of reserved instances initially based on normal distribution (note we will accept only those numbers generated in range [0, 1]). The averages for each services are provided in the following:

TABLE 4.1: Parameters of Demand Generation (Dataset 1)

Demand Type	*Regular*	*New Year's Holiday*	Weekends
μ_r	50	15	10
μ_o	60	20	12
μ_s	300	200	150
μ_u	0.50	0.45	0.40
σ_u	0.15	0.15	0.15

4.2.2 Dataset 2

This dataset is similar to the case of dataset 1 without considering the fluctuation of demand over time. In other words, the customers arrivals are based on a constant rate. We evaluate the effect of demands when their averages are constant over the horizon rather than interchangeable. Thus, their parameters of this dataset will be:

TABLE 4.2: Parameters of Demand Generation (Dataset 2)

Demand Type	*Regular*	*New Year's Holiday*	Weekends
μ_r	50	—	—
μ_o	60	—	—
μ_s	300	—	—
μ_u	0.50	—	—
σ_u	0.15	—	—

4.2.3 Dataset 3

This dataset considers the case where arrival rates are interchangeable and will fluctuate over the horizon. They will be at their highest peak in regular days and in holidays the rate will decrease. Moreover, In weekends they will reach their lowest level. This is the case where most of the cloud's capacity is filled by on-demand instances rather than reserved.

TABLE 4.3: Parameters of Demand Generation (Dataset 3)

Demand Type	*Regular*	*New Year's Holiday*	Weekends
μ_r	20	12	8
μ_o	150	40	25
μ_s	300	200	150
μ_u	0.50	0.45	0.40
σ_u	0.15	0.15	0.15

In the next section we will provide the results of the simulation runs for each data set in discuss them.

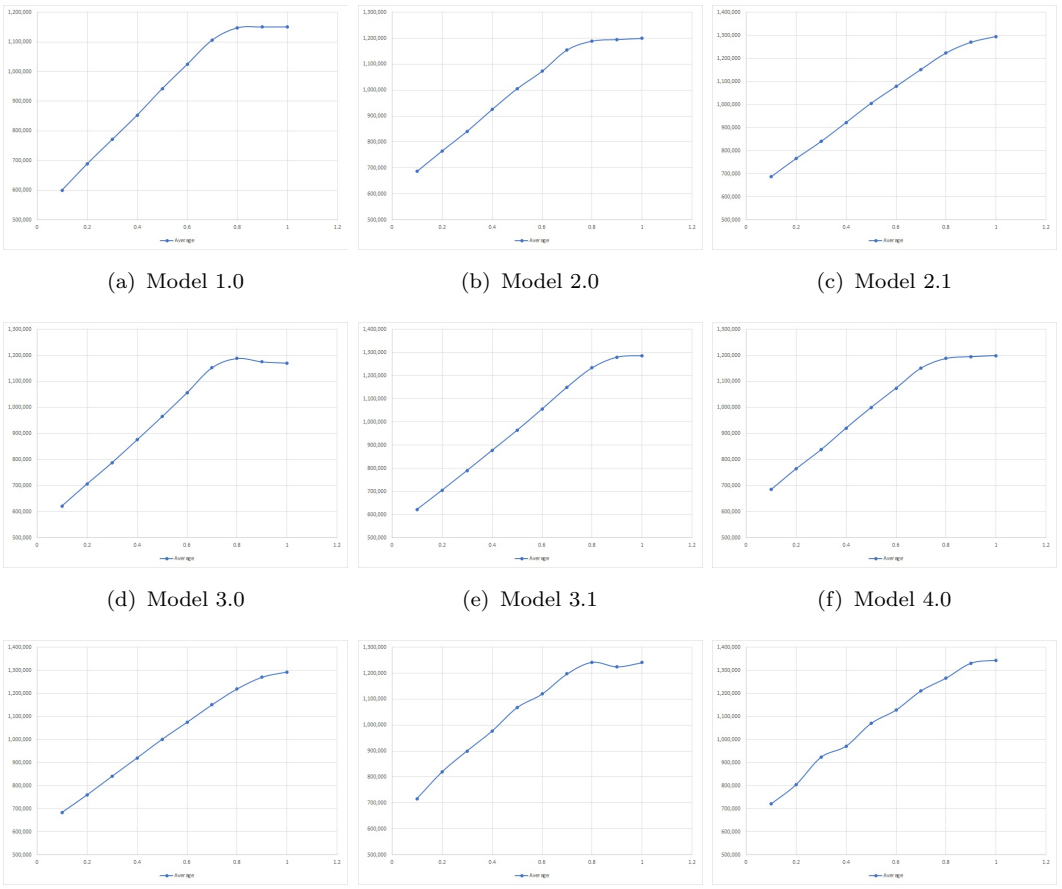
4.3 Results

The output of the simulation algorithms are presented in this section, we categorized them by two layers; (i) dataset, and (ii) capacity, to discuss the optimal ρ out of 10 options (0.1, 0.2, 0.3, ..., 1.0). Then, we will evaluate the revenue of each pricing policy. We have simulated each scenario based on their parameters 5 times, and computed the 95% confidence intervals for the revenue accordingly.

4.3.1 Revenue in Dataset 1

4.3.1.1 Capacity 1500

At first, we have the graph of average revenue of each pricing model for different values of ρ (figure 4.1). We see that all the graphs are increasing at a



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.1: Total Revenue for Capacity 1500 and Dataset 1

rather constant positive rate initially; however, in the final points this rate declines and even becomes negative in a few cases. The reason for this behavior of revenue based on ρ is that at small values of ρ the capacity is not fully occupied. So, by increasing the value of ρ, the revenue will increase. Nevertheless, at values higher than 0.8 the cloud cannot receive more instances due to its capacity. In some cases, as in figure 4.1 (d) there is a maximum point and then the revenue will decrease. This is due to the fact that other alternatives produce higher revenues. Figure 4.1 (d) and (h) suggest the importance of finding the optimal value for ρ.

By having the maximum revenues for each pricing model (which are represented in table 4.4). To accurately compare the revenues of different model we have calculated the *95% confidence interval* of their revenues and found their lower and upper bounds.

Table 4.4: Maximum Revenues for Capacity 1500 - Dataset 1

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1500	1.0	1,147,024	1,150,201	1,153,379
Model 2.0	1500	1.0	1,196,921	1,199,473	1,202,024
Model 2.1	1500	1.0	1,290,628	1,294,301	1,297,975
Model 3.0	1500	0.8	1,176,421	1,188,373	1,200,325
Model 3.1	1500	1.0	1,274,902	1,285,681	1,296,460
Model 4.0	1500	1.0	1,195,868	1,198,696	1,201,524
Model 4.1	1500	1.0	1,289,480	1,292,677	1,295,874
Model 4.2	1500	0.8	1,230,364	1,242,586	1,254,807
Model 4.3	1500	1.0	1,336,606	1,342,973	1,349,339

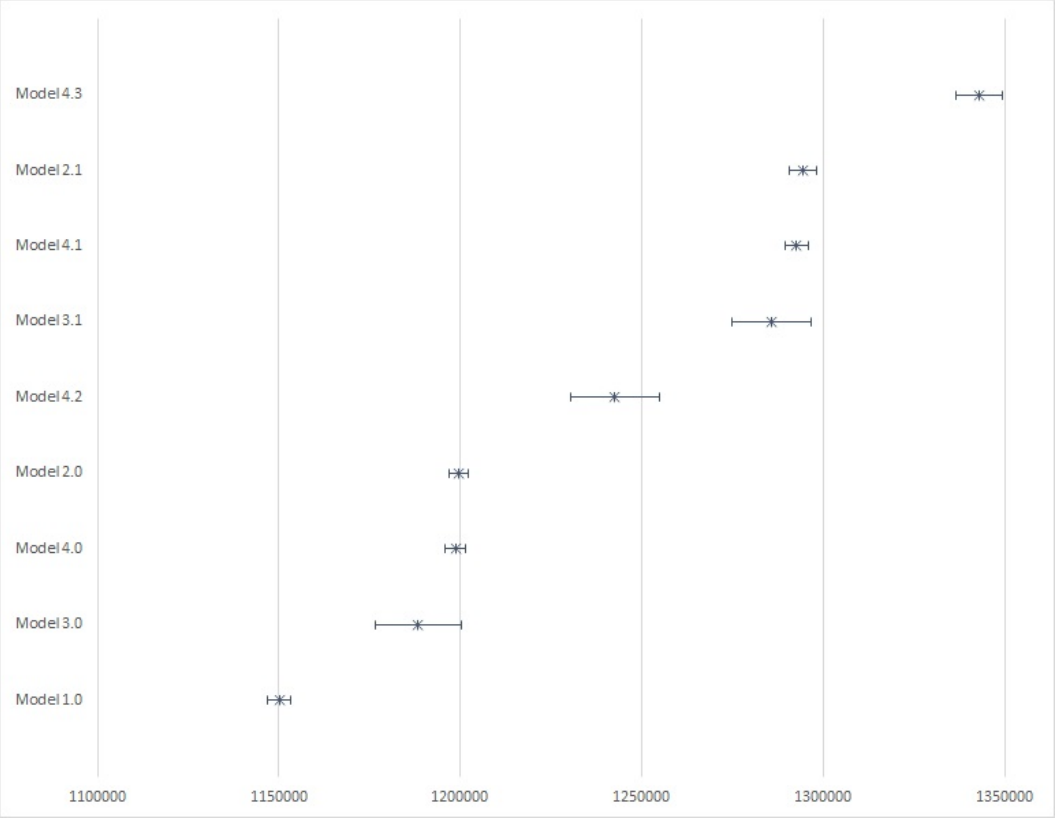


Figure 4.2: Revenue Comparison of Models
Capacity 1500, Dataset 1

We used graphic tools to compare the pricing policies (figure 4.2). We have ordered the models in based on their average revenues. The left cap of each interval indicates the lower bound, whereas its right cap is the upper bound, and $(*)$ sign in the middle indicates average value of each revenue over the 5 runs. Figure 4.2 shows that model 4.3 produces the highest revenue, and models 3.0, 4.0, and 2.0 have the least revenues (since their confidence intervals overlap with each other

we cannot differentiate between them). We see that model 4.3 has a 13% increase in revenue compared to model 3.0, which suggests the significance of selecting a policy for the service provider. Model 4.3 employs the queuing mechanism for on-demand instances and model 4.2 is without it, where the difference in their revenue is 8.1%. Moreover, we can see that by exploiting the adaptable threshold (model 4.3 vs 4.1) the revenue is increased by 3.9% (the average threshold for model 4.3 is \$0.555). It is logical in this case (capacity 1500), for there is capacity shortage and the threshold increases. Model 3.1 is similar to 4.3 with a slight difference, where in selection of spot instances the one with highest bid is selected rather than highest expected revenue. The difference in their revenues is 4.5%. We have summarized these differences in the table below.

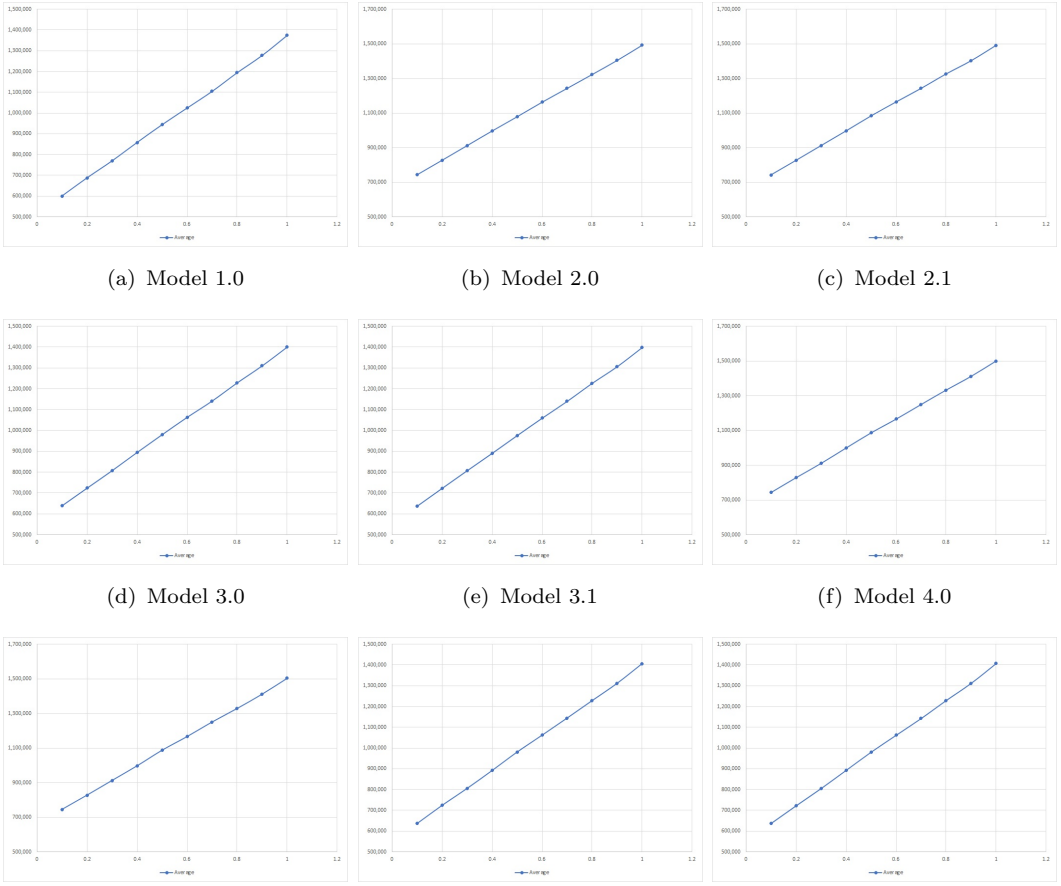
TABLE 4.5: Optimal Model's Revenue vs. Other Models
Capacity 1500, Dataset 1

Optimal Model (3.1) vs.	*Difference in Revenue (%)*
4.3	13.1
4.2	8.1
4.1	3.9
3.1	4.5

4.3.1.2 Capacity 2500

In this case, we investigate the dataset 1 where we have sufficient capacity and there will not be inadequacy in the cloud's facilities. Figure 4.3, illustrates the effect of ρ on revenue for each pricing policy. The rate of increase in revenue for each model is constant in this case. This trend is the result of capacity sufficiency, the more instances the CSP receives the higher its revenue will be. In other words, when the capacity is not a constraint for the system, optimal value for ρ will be equal to 1. Thus, the maximum revenue of each pricing policy for this capacity will be as displayed in table 4.6.

The corresponding graph to this table is figure 4.4. As expected, the queuing policy and selection of bids based on the two criteria presented in the models, are irrelevant in this case. Since, the capacity is sufficient none of the on-demand



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.3: Total Revenue for Capacity 2500 and Dataset 1

TABLE 4.6: Maximum Revenues for Capacity 2500 - Dataset 1

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	2500	1.0	1,367,333	1,374,543	1,381,753
Model 2.0	2500	1.0	1,482,449	1,492,614	1,502,780
Model 2.1	2500	1.0	1,482,434	1,491,188	1,499,941
Model 3.0	2500	1.0	1,391,342	1,399,911	1,408,481
Model 3.1	2500	1.0	1,388,418	1,398,522	1,408,626
Model 4.0	2500	1.0	1,491,606	1,500,075	1,508,545
Model 4.1	2500	1.0	1,494,551	1,503,314	1,512,076
Model 4.2	2500	1.0	1,396,665	1,405,819	1,414,972
Model 4.3	2500	1.0	1,399,605	1,407,120	1,414,635

instances will be rejected by the cloud, and there will be no queue for on-demand customers. In addition, all of the spot instances will be received to the cloud, so the concern of selecting one of the two policies for bid selection becomes redundant. However, the criterion which causes the difference in figure 4.4 is the threshold

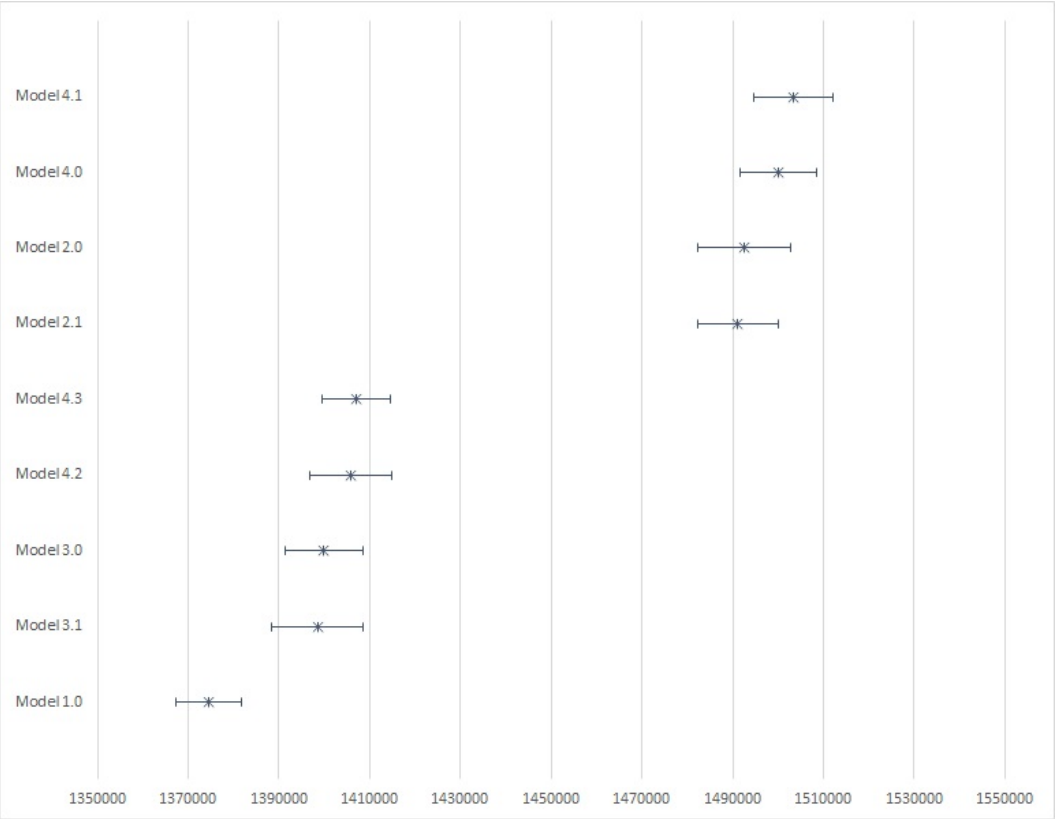
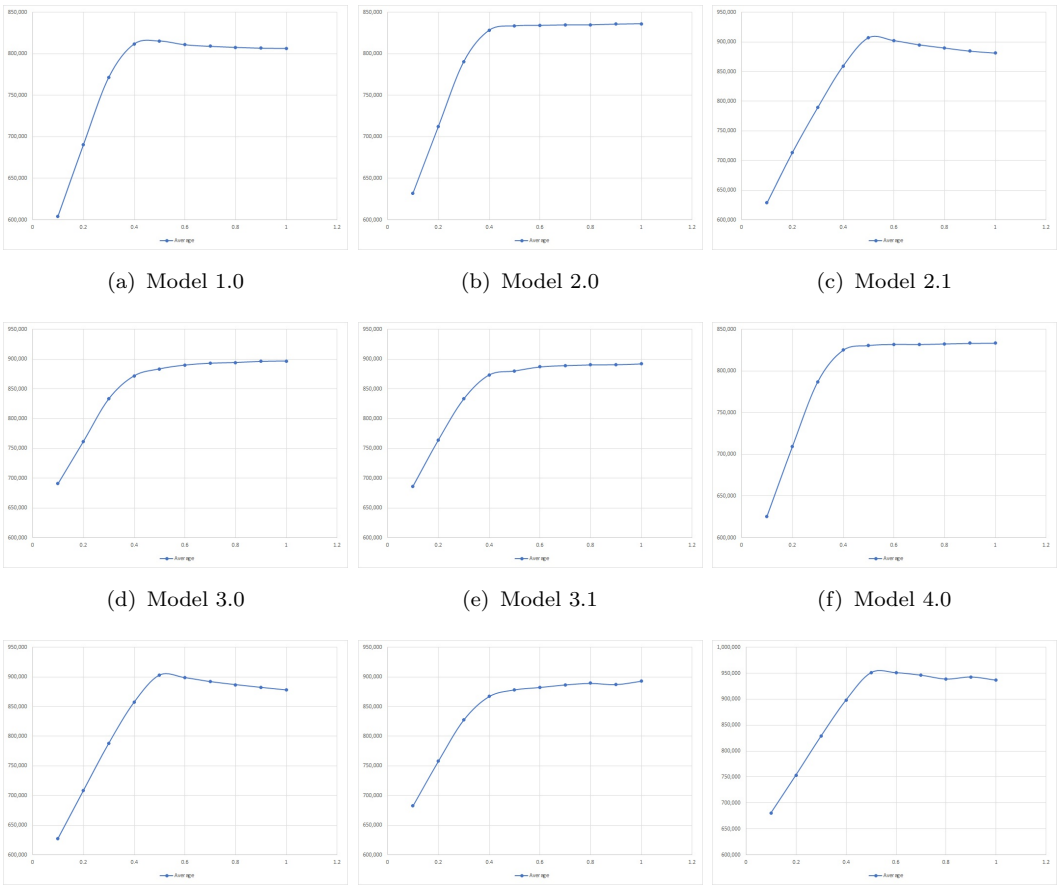


FIGURE 4.4: Revenues Comparison of Models
Capacity 2500, Dataset 1

announcement to the customers. In the models with high revenue (e.g. 2.0) a constant threshold is published to the users of spot instances, while in the other models (e.g. 3.0) the threshold announced to the public is based on the previous period bids accepted to the system. Since all of the bids are received to the cloud in this case, the threshold's value declines over time (the average threshold in this case is \$0.101). The difference between the revenue of model 2.0 versus 3.0 is 6.6%, so we can conclude that the CSP should choose a policy with constant threshold. Note that this conclusion holds for all datasets with abundant capacity (e.g. datasets 2 and 3 with capacity 2500).

4.3.1.3 Capacity 1000

This case examines the factors affecting revenue where capacity is excessively low. The simulation runs under this situation manifest that the capacity becomes saturated with lower values of ρ (figure 4.5). However, the revenue patterns in this



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.5: Total Revenue for Capacity 1000 and Dataset 1

case can be categorized into two classes. Class 1, happens in models such as 2.1 where maximum revenue is achieved when around 50% of the reserved customers are accepted into the system. The cause of this phenomenon is that in these cases the revenue generated by accepting more of reserved users is less than the loss in revenue of rejecting on-demand instances. On the other hand, class two demonstrate that revenue will increase on an extreme rate for values of ρ less than 0.4, while for values greater than that this rate will decline drastically. This indicates that the after a certain ρ there is a trade off between revenue gain of reserved instances and revenue loss on-demand customers. Table 4.7, manifests the optimal revenues for each model.

In order to clarify the differences of revenue in each model we have provided a graph of their average revenue and confidence intervals (figure 4.6). We see that the pattern of income generated by different models in this section is quite similar

TABLE 4.7: Maximum Revenues for Capacity 1000 - Dataset 1

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1000	0.5	813,097	815,481	817,864
Model 2.0	1000	1.0	833,787	836,137	838,487
Model 2.1	1000	0.5	902,299	906,902	911,505
Model 3.0	1000	1.0	894,535	896,562	898,588
Model 3.1	1000	1.0	889,887	891,965	894,042
Model 4.0	1000	1.0	831,420	833,502	835,585
Model 4.1	1000	0.5	898,55	902,884	907,215
Model 4.2	1000	1.0	883,670	892,937	902,204
Model 4.3	1000	0.5	947,286	951,276	955,266

to the case where capacity is 1500. Model 4.3 generates the highest revenue by a measurable excess.

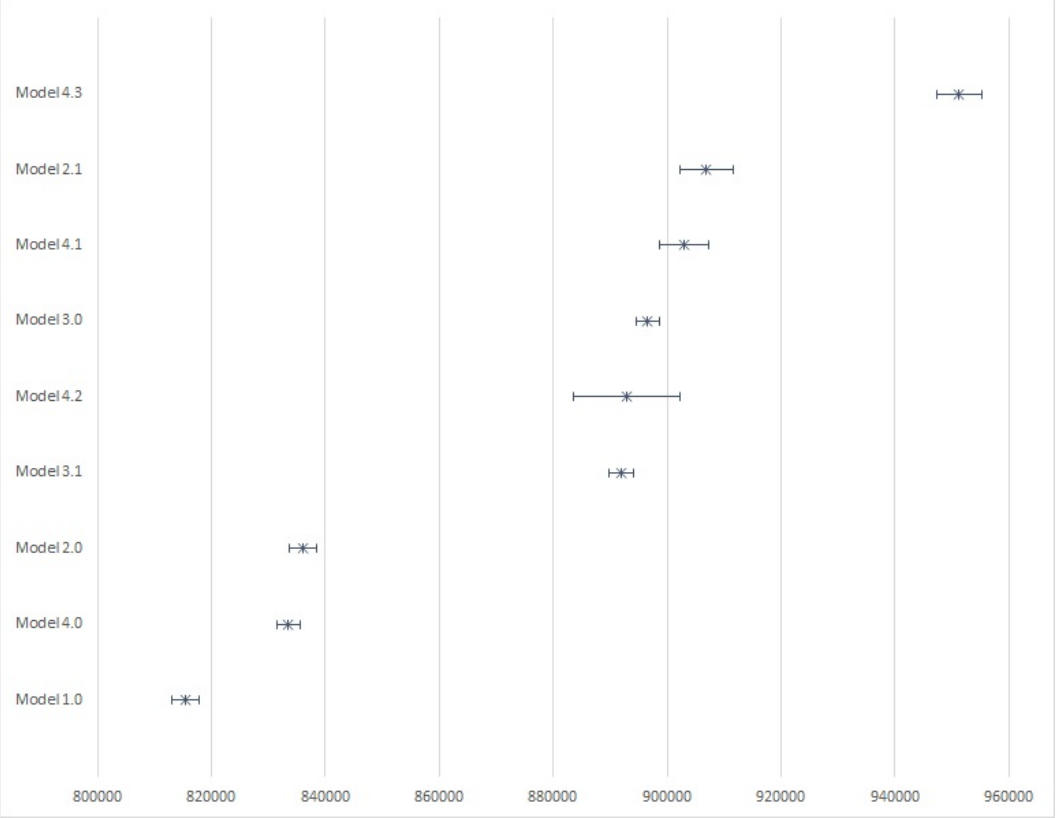


FIGURE 4.6: Revenues Comparison of Models
Capacity 1000, Dataset 1

As a means to demonstrate the sensitivity of model selection by the CSP, we have provided the list of revenue growth by choosing this model compared to other policies (table 4.8). By incorporating any of the policies suggested in this study

Table 4.8: Optimal Model's Revenue vs. Other Models
Capacity 1000, Dataset 1

Optimal Model (4.3) vs.	*Difference in Revenue (%)*
4.2	6.5
4.1	5.4
3.1	6.7
2.0	13.8

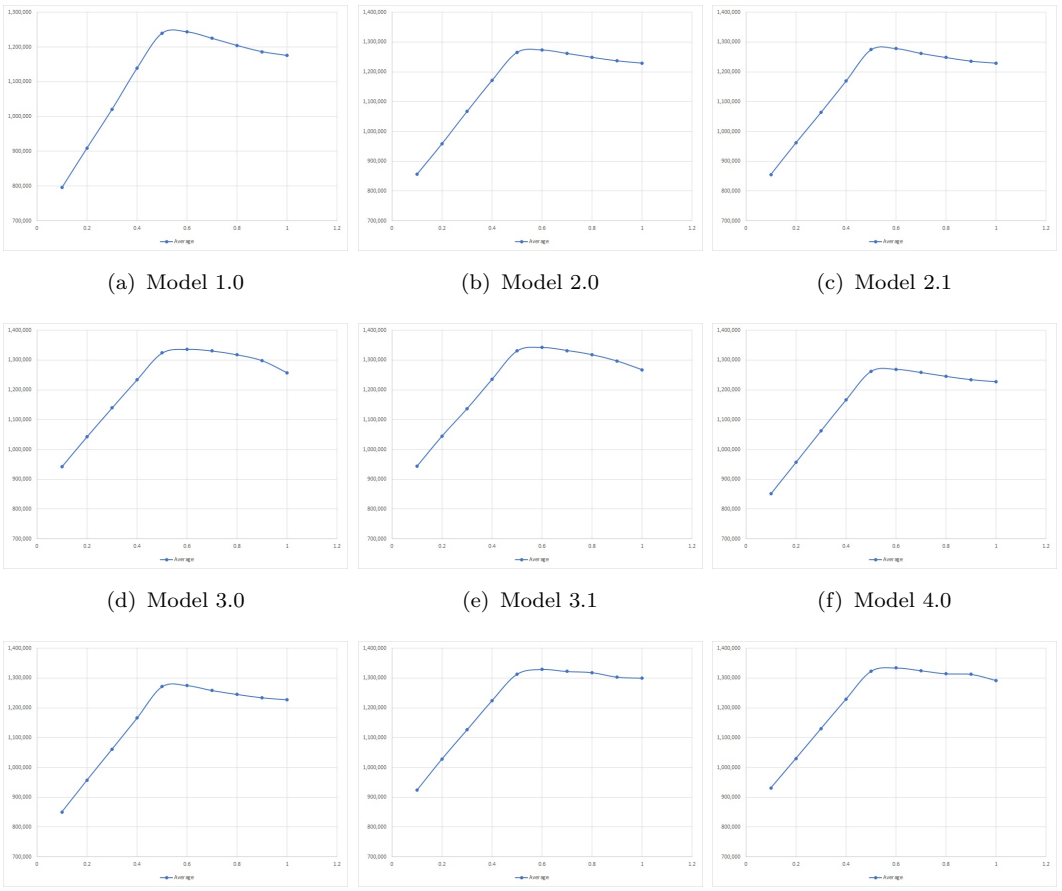
the CSP will acquire a revenue growth of around 6%, which means all of them have the same level of importance for the cloud. Moreover if the service provider chooses the primitive model 2.0 will generate 13.8% less revenue compared to the optimal model.

4.3.2 Revenue in Dataset 2

In this dataset we intend to investigate the effect of arrival rates in dataset 1 to be constant on the total revenue. As mentioned before the results of this demand pattern for capacity 2500 are similar to that of dataset 1, so we only examine the capacities 1500 and 1000 in for this dataset.

4.3.2.1 Capacity 1500

Initially, we inspect every model's revenue according to the change in ρ. Figure 4.7 illustrates the revenue for each pricing plan based on the proportion of reserved instances accepted to the system. This dataset represents a cloud where the average demand for on-demand instances is less than that of reserved instances. We see in each figure that the revenue will increase by accepting a larger portion of reserved customers at first. However, it will arrive at a peak where the revenue is maximum and decline afterwards. This phenomenon is the result of two facts; first, at optimal ρ the cloud becomes full and prior to that we have empty capacity. Two, the revenue of on-demand instances is higher than that of reserved instances in general.



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.7: Total Revenue for Capacity 1500 and Dataset 2

Mainly, a ρ equal to 0.6 yields the highest revenue in this dataset. we have provided the list maximum revenues in table 4.9 for all pricing models. However,

TABLE 4.9: Maximum Revenues for Capacity 1500 - Dataset 2

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1500	1.0	1,239,096	1,243,509	1,247,922
Model 2.0	1500	1.0	1,269,961	1,273,489	1,277,018
Model 2.1	1500	1.0	1,274,823	1,278,556	1,282,289
Model 3.0	1500	0.8	1,333,438	1,336,433	1,339,427
Model 3.1	1500	1.0	1,337,661	1,341,913	1,346,165
Model 4.0	1500	1.0	1,265,580	1,269,006	1,272,431
Model 4.1	1500	1.0	1,270,940	1,275,070	1,279,200
Model 4.2	1500	0.8	1,322,593	1,329,253	1,335,913
Model 4.3	1500	1.0	1,330,331	1,333,916	1,337,502

we use the graph in figure 4.8 to facilitate the comparison between pricing scheme. In this case, we observe that the queuing policy does not affect the total revenue

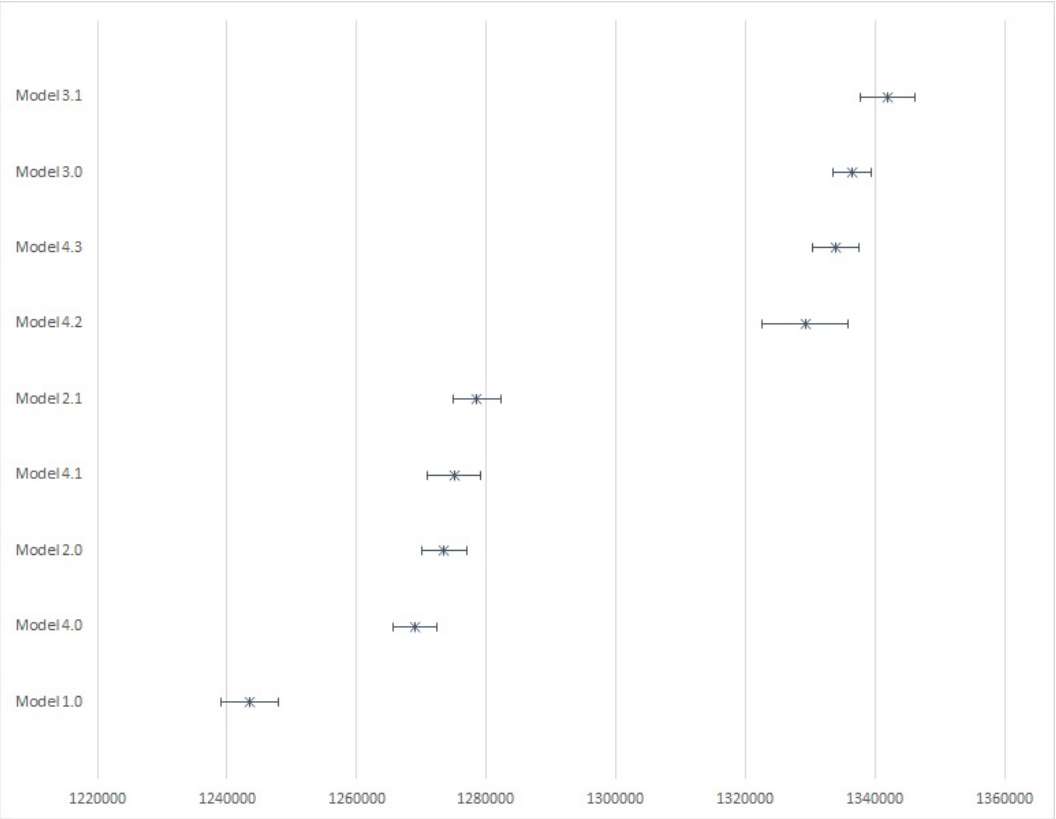


FIGURE 4.8: Revenues Comparison of Models
Capacity 1500, Dataset 2

significantly, i.e. the confidence interval of the models with and without this policy intersect. For example, the confidence interval for model 3.1 which represents the queuing model and 3.0 where there is no queue intersect with each other. So, we cannot state that model 3.1 yields a higher revenue. The logic of queuing model is when the system has capacity shortage in general; while, there are periods where the cloud will have vacant capacity to due a decline demand (e.g. dataset 1). Nevertheless, this is not the case for this dataset, which will cause queuing policy to become ineffectual.

Moreover, as the cloud is mostly overloaded in the horizon, there are little number of spot instances accepted to the system. In this regard, the difference in bid selection policies becomes insignificant. This conclusion is validated by figure 4.6, for example we cannot state any difference in the revenue of model 4.2 and 3.0. However, the adaptable threshold will produce higher revenues. For instance, model 3.1 yields 5% more revenue rather than model 2.1 on average. In this case

the service provider should choose a policy with conformable threshold to ensure a higher revenue. Table 4.10 demonstrates the difference in revenue based on model.

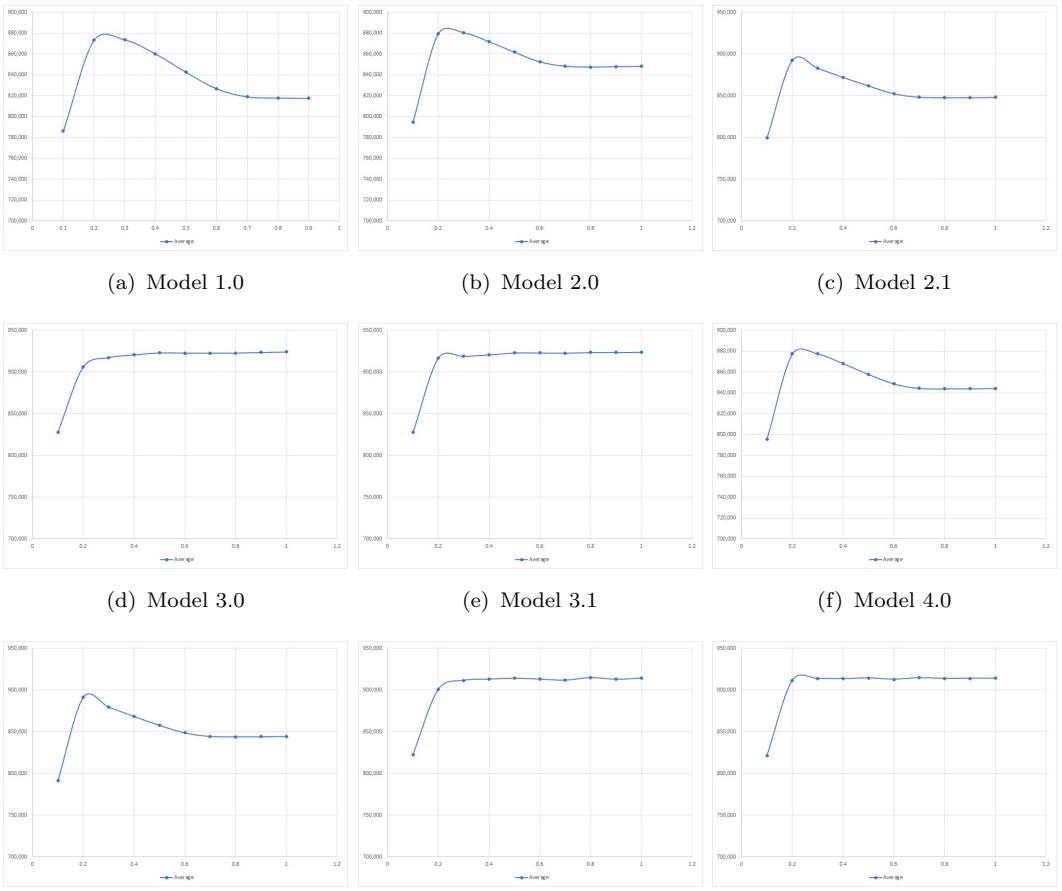
TABLE 4.10: Optimal Model's Revenue vs. Other Models
Capacity 1500, Dataset 2

Optimal Model (3.1) vs.	*Difference in Revenue (%)*
3.0	NaN
4.3	NaN
2.1	5

4.3.2.2 Capacity 1000

This case examines the factors affecting revenue where capacity is excessively low. The simulation runs under this situation manifest a new pattern for ρ selection in order to achieve higher revenues (figure 4.9). By changing ρ two patterns are derived for revenue. First, the income starts at its lowest worth for smallest value of ρ and by increasing it to 0.2 the CSP will receive the highest revenue possible. Increasing ρ to a greater extend will reduce the income (e.g. model 4.1). Thus, the value 0.2 provides the marginal revenue, this is due to the income of each demand type. In other words, the revenue yielded by accepting more reserved customer is less than the revenue loss of rejected on-demand users. The second case is observed in models such as 4.2 where revenue will reach its higher value after accepting 20% of reserved instances and stay on the relatively same value by accepting more reserved instances. The former case happens in models when there is no adaptable threshold, while the latter is caused by the adaptable threshold policy. We can deduce that the revenue obtained for spot instances by these models compensate for the revenue loss of on-demand users. The optimal values for ρ and its revenue for each model are manifested in table 4.11.

The statistics of revenue for different models indicate that queuing policy (which is applied to models such as 4.3) has the least effect in this case. The cause of this phenomenon is that queue cannot form, since rate of demand is constant and capacity does not become available at any period. In other words, queuing



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.9: Total Revenue for Capacity 1000 and Dataset 2

policy is irrelevant under these circumstances as the confidence interval of revenue for models with this policy versus those without it overlap. Moreover, the most critical policy for revenue growth in this case is adaptable threshold (e.g. model 3.0 and 4.2). These finding are observed in figure 4.10 as well. Nevertheless,

TABLE 4.11: Maximum Revenues for Capacity 1000 - Dataset 2

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1000	0.3	871,843	873,633	875,423
Model 2.0	1000	0.2	878,864	880,488	882,111
Model 2.1	1000	0.2	888,741	892,396	896,051
Model 3.0	1000	1.0	921,808	924,051	926,294
Model 3.1	1000	1.0	921,490	923,873	926,255
Model 4.0	1000	0.3	875,697	877,393	879,089
Model 4.1	1000	0.3	878,438	879,681	880,924
Model 4.2	1000	0.8	910,421	914,870	919,320
Model 4.3	1000	0.7	910,550	914,544	918,537

the phenomenon that models using expected revenue for spot selection generate less income than models which choose instances based on highest bid is the most intriguing finding of this case. Although the difference in revenue of these models are small-scale, the cause of it requires to be detected. This difference in revenue

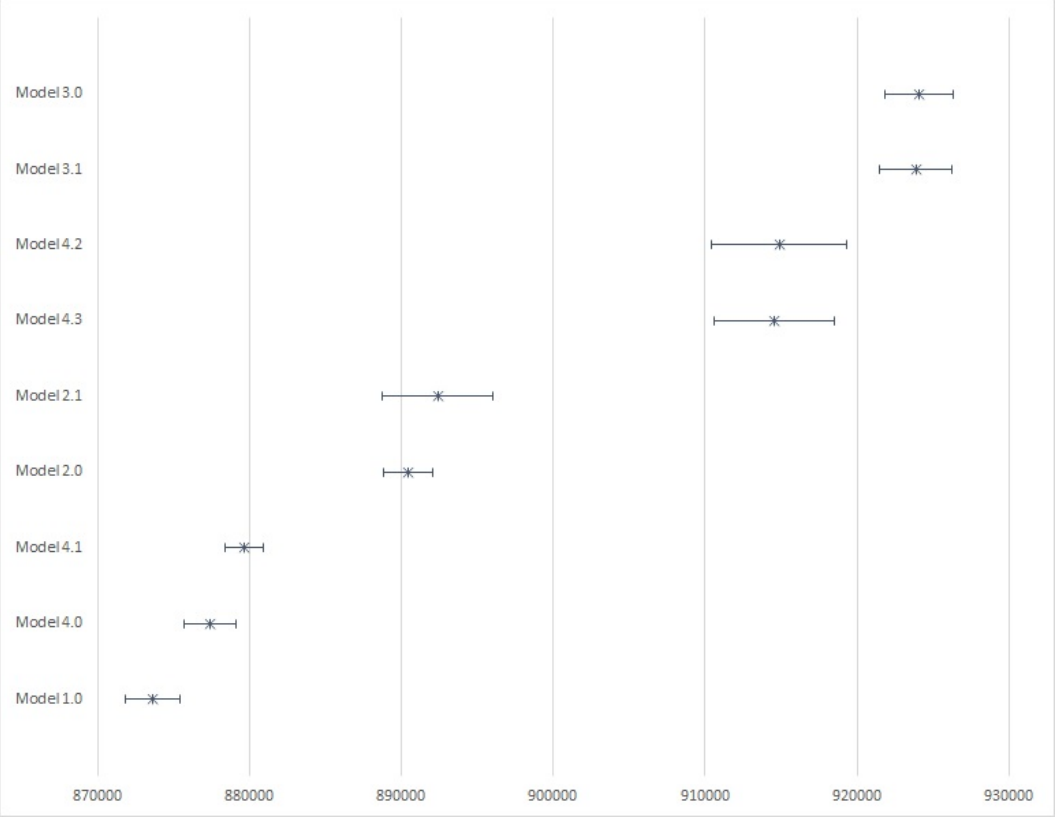


FIGURE 4.10: Revenues Comparison of Models
Capacity 1000, Dataset 2

is 1% for model 3.0 versus 4.2. For cases where the cloud is faced with capacity shortage, the probability of kicking a spot instance out of the system increases. This causes instances with higher bids more appealing to the service provider, as they will most likely be dropped out of the system before their service duration is complete.

The difference in revenues of distinct models are relatively minor, due to the lack of flexibility when capacity is extremely low. These differences are manifested by percentages in table 4.12. In general, adaptable threshold will result in the highest revenue growth when rate of demand is constant and there is capacity

Table 4.12: Optimal Model's Revenue vs. Other Models
Capacity 1000, Dataset 2

Optimal Model (3.0) vs.	*Difference in Revenue (%)*
3.1	NaN
4.2	1
2.0	4.9

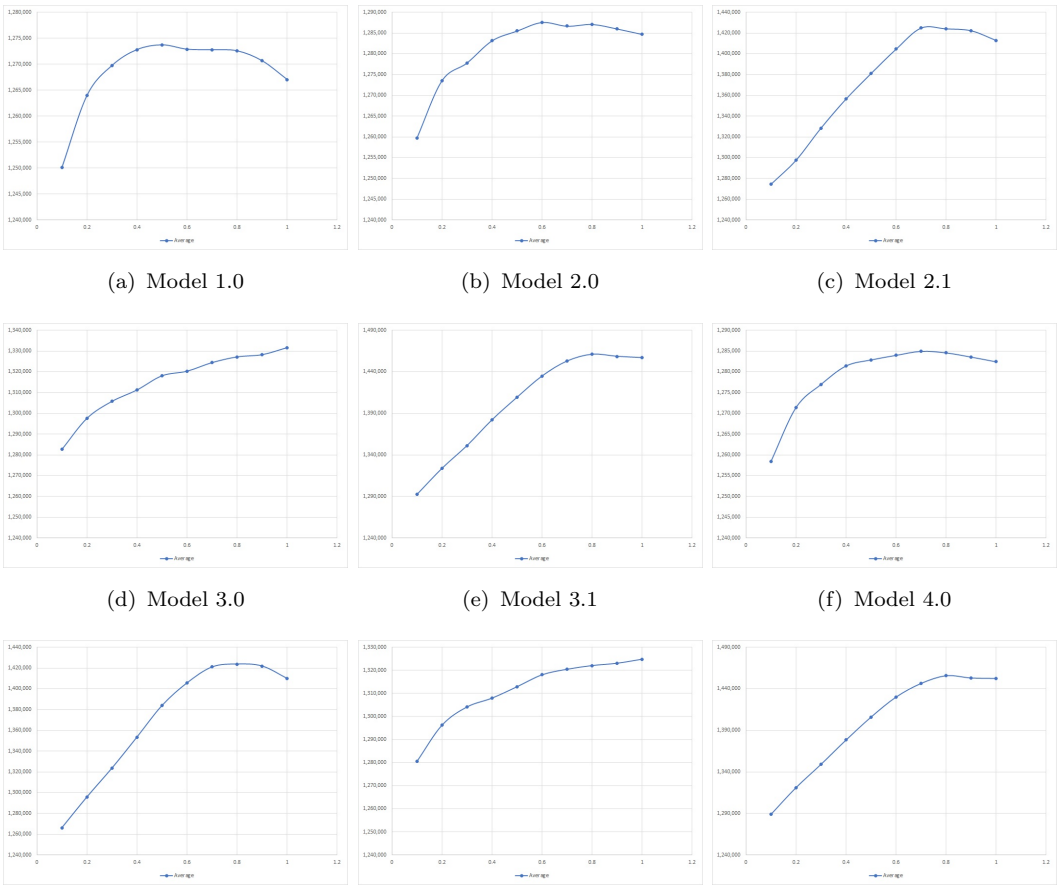
shortage. This can be concluded by observing the findings of both cases with capacity 1500 and 1000 in this dataset.

4.3.3 Revenue in Dataset 3

This dataset represents a case where demand will fluctuate over time on average, and the number of on-demand instances is higher than reserved ones. As for the previous case we only discuss the results for capacity 1500 and 1000 where there is a shortage of resources. Because, the policies' behavior is the same when capacity of the cloud is abundant for the customers.

4.3.3.1 Capacity 1500

Figure 4.11 demonstrates the effect of ρ on revenue of the cloud. Obviously, for small values of ρ if the service provider accepts more proportion of reserved instances, revenue of the cloud will increase. In other words, for small values of ρ the cloud's capacity is not fully occupied, so by accepting more reserved instances the revenue will increase. However, for all models except model 3.0 and 4.2 the revenue will be at its highest when ρ is around to 0.7. This indicates that these values of ρ are the trade-off points where other sources of income become more profitable compared to accepting more reserved instances. This analysis does not hold true for models 3.0 and 4.2 which are very similar in term of their policies. The difference between these two model is that the former accept spot customers based on bids, while the latter's criterion for this selection is expected revenue.



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.11: Total Revenue for Capacity 1500 and Dataset 3

In these cases, the cloud will receive more revenue by accepting as much reserved customers as it can.

The cause of this type behavior is that in pricing policies where there is a revenue for queue and threshold is constant the income of other resources (e.g. on-demand revenue) compared to reserved instances become more critical. On the other hand, for cases where threshold is adaptable and there is no income for rejected on-demand instances, accepting more reserved customers will yield higher values of revenue. The summary of best values for ρ and their revenue is provided in table 4.13.

For simplicity, we use figure 4.12 to compare the pricing models. The confidence interval of policies with highest bid selection versus best expected revenue option (e.g. 3.1 vs. 4.3) for spot customers intersect in this dataset, while this

TABLE 4.13: Maximum Revenues for Capacity 1500 - Dataset 3

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1500	0.5	1,271,512	1,273,703	1,275,895
Model 2.0	1500	1.0	1,286,187	1,287,567	1,288,946
Model 2.1	1500	1.0	1,418,422	1,424,921	1,431,419
Model 3.0	1500	0.8	1,330,120	1,331,662	1,333,204
Model 3.1	1500	1.0	1,458,789	1,460,916	1,463,043
Model 4.0	1500	1.0	1,282,894	1,284,935	1,286,975
Model 4.1	1500	1.0	1,419,465	1,423,791	1,428,117
Model 4.2	1500	0.8	1,321,663	1,323,081	1,324,499
Model 4.3	1500	1.0	1,452,603	1,455,661	1,458,720

was not the case in data 1. This difference in behavior is because in dataset 1 we had more reserved customers rather than this case, and the difference in income of these two policies become negligible in this dataset. nonetheless, models with adaptable threshold and queuing policy yield more income rather than the other models. Table 4.10 illustrates these differences in revenue in terms of percentage for model 3.1 on average.

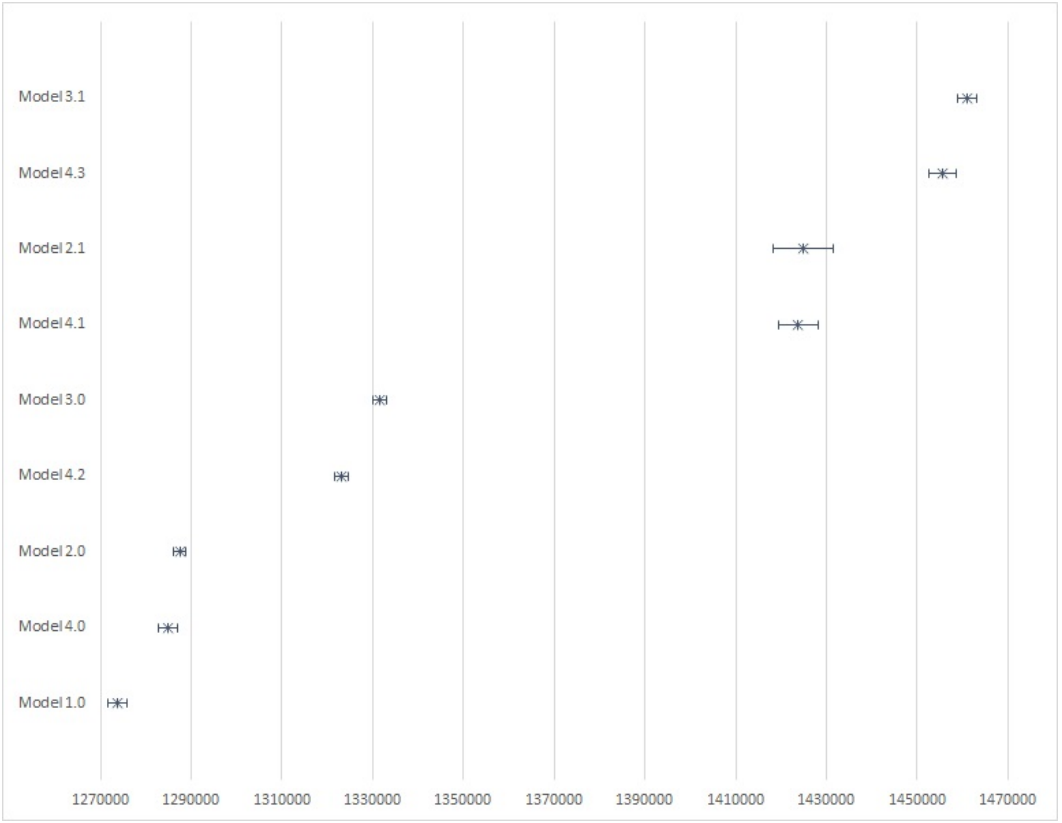


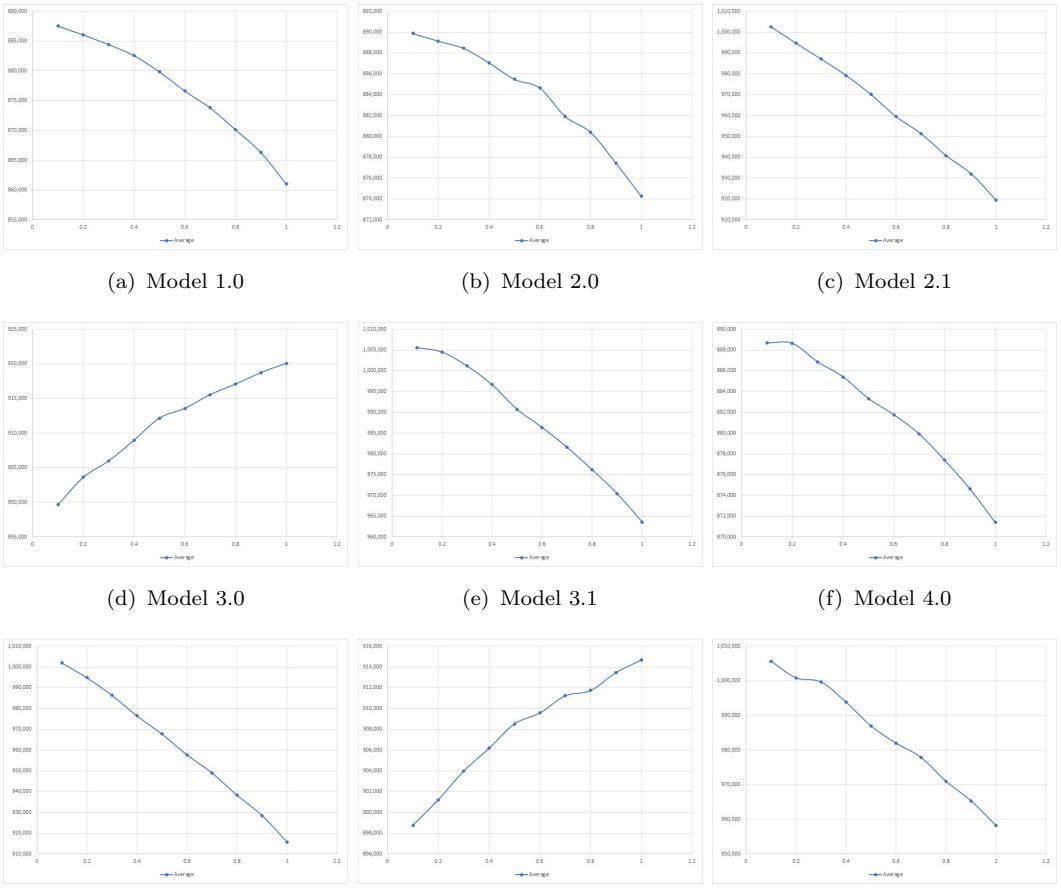
FIGURE 4.12: Revenues Comparison of Models
Capacity 1500, Dataset 3

By way of conclusion, the service provider should choose a policy with queue and adaptable threshold. In addition, queuing scheme is more critical for revenue growth rather than adaptable threshold policy in this case.

TABLE 4.14: Optimal Model's Revenue vs. Other Models
Capacity 1500, Dataset 3

Optimal Model (3.1) vs.	*Difference in Revenue (%)*
4.3	NaN
3.0	9.7
2.1	2.5

4.3.3.2 Capacity 1000



(a) Model 1.0 (b) Model 2.0 (c) Model 2.1

(d) Model 3.0 (e) Model 3.1 (f) Model 4.0

(g) Model 4.1 (h) Model 4.2 (i) Model 4.3

FIGURE 4.13: Total Revenue for Capacity 1000 and Dataset 3

In this case the findings of revenue based on ρ are dissimilar from other cases. The income in all models except model 3.0 and 4.2 has a negative correlation

with proportion of reserved customers accepted to the system. This decreasing pattern happens due to capacity shortage, resulting in saturation of the cloud with only 10% of the reserved customers. As demand for on-demand instances is high in this case and its revenue is higher than reserved ones, the income will decline by accepting more users for reservation plan. However, model 3.0 and 4.2 indicate a positive correlation between the value of ρ and revenue. The cause of this increase is because the gain from spot instances using adaptable threshold policy is more than revenue loss of on-demand users. Although this increasing pattern happens for these two model, the revenue-ρ graph has a decreasing curve for other models employing adaptable threshold (e.g. model 3.1). The reason for their negative correlation is that revenue of on-demand customers from the queue prevail the income generated from spot users.

The optimal revenue and its confidence interval for all models are gathered in table 4.15 with their ρ values. In addition, they are illustrated by the following chart (figure 4.14). It corroborates the findings from the previous paragraph

Table 4.15: Maximum Revenues for Capacity 1000 - Dataset 3

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1000	0.1	886,387	887,527	888,667
Model 2.0	1000	0.1	888,954	889,883	890,812
Model 2.1	1000	0.1	1,001,237	1,002,587	1,003,936
Model 3.0	1000	1.0	914,909	916,060	917,211
Model 3.1	1000	0.1	1,002,322	1,005,574	1,008,826
Model 4.0	1000	0.1	887,485	888,664	889,844
Model 4.1	1000	0.1	999,663	1,001,963	1,004,263
Model 4.2	1000	1.0	912,768	914,675	916,583
Model 4.3	1000	0.1	1,002,741	1,005,640	1,008,539

regarding the effect of queuing and adaptable policy. We see that models with constant threshold and no queue have the least revenue. Moreover, incorporating adaptable threshold will result in a growth for income, though it is minute (model 3.0 generates 2.9% more revenue than 2.0). However, by employing the queuing policy revenue will increase more drastically. The cause of this extreme growth is that demand for on-demand services is very high compared to the capacity, and revenue generated by this service is higher than the other services. Thus, by

utilizing the queuing policy revenue of the cloud will increase significantly. For example, income of model 4.3 is 10% higher than revenue of model 4.2.

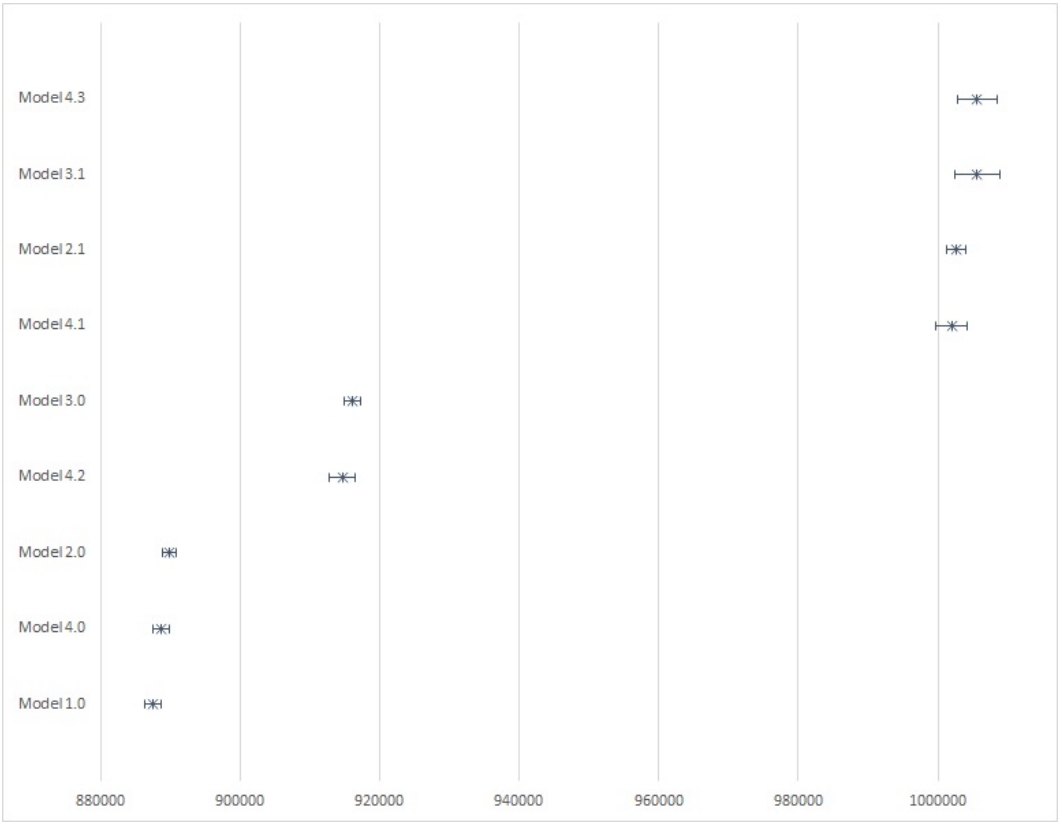


FIGURE 4.14: Revenues Comparison of Models
Capacity 1000, Dataset 3

Table 4.16 demonstrates the difference amongst models, in order to clarify the significance of employing different policies. In conclusion, as a CSP the most crucial policy is queuing scheme, where demand for on-demand instances is high and there is extreme capacity shortage.

TABLE 4.16: Optimal Model's Revenue vs. Other Models
Capacity 1000, Dataset 3

Optimal Model (4.3) vs.	*Difference in Revenue (%)*
3.1	NaN
2.1	NaN
4.1	NaN
4.2	10
4.0	13.2

In the following chapter we will provide a summary for this study, discuss the conclusion of our findings and provide a groundwork for future studies in this field.

Chapter 5

Conclusion and Future Work

Large-scale data centers and the cloud computing services they provide are rapidly becoming more widespread in the IT world. Service providers such as Amazon EC2 and Google Cloud Platform as the largest data centers are receiving many users from around world. Due to the nature and scale of revenue in these service providers, utilizing revenue management techniques on these clouds has drawn many researchers' attention to itself. A data center provides three main type of services - SaaS, PaaS, and IaaS - where the focus of this study was on IaaS's revenue. Customers of IaaS can choose three pricing plans based the quality of service that they require. These plans are subscription, on-demand, and spot instances. Recent literature have mainly focused on optimizing and pricing of spot instances' revenue, and do not consider other plans in their calculations. However, Toosi *et al.* proposed a model in 2015 which considers all the pricing plans of the cloud in their analysis, and find the best assignment of resources to demand accordingly. Despite their state of the art research, their model lacked realistic assumptions and pricing options for on-demand and spot instances.

In this thesis, we have provided a simulation analysis, over a finite time horizon, evaluating three assignment policies and their combinations. We investigated the effects of (i) constant threshold versus adaptable to market and capacity threshold, (ii) introducing a queue for rejected on-demand instances (with the incentive of lower instance price), and (iii) two criterion for spot selection (based on

bid versus expected revenue). We implemented each combination of these policies in Python, and analyzed their outcome. However, to due lack of information about demand, we have generated three sets of data to test our models. The results of this implementation suggest the importance of policy selection in a system with low capacity. In other words, the revenue of a cloud service provider is sensitive to policy selection when there is capacity shortage. Mainly for cases with deficiency of resources and fluctuation of demand over time, revenue is higher in value when we incorporate the three policies suggested by this study. On the other hand, when the demand average is constant only adaptable threshold will increase the revenue. However, when capacity is sufficient for demand these policies does not affect the revenue except adaptable threshold will cause income of the clout to decline, thus, choosing a constant threshold is more beneficial to the service provider.

There are several lines of research arising from this study which are required to be followed. First, incorporating the costs relating to data center and resource allocation can effect the decision of cloud provider gravely. Second, another line of research can be considering the demand to be unknown at the beginning of the time horizon and solving the problem by optimization. Finally, combining a dynamic pricing algorithm for instances offered by the service provider to regulate the demand.

Appendix A

Parameters and Variables

In this appendix we will provide all the parameters and variables which are used with throughout this thesis with their description.

$\boldsymbol{\rho}$	The proportion of reserved demands considered for acceptance
$\boldsymbol{C}$	Capacity of the cloud
$\boldsymbol{\tau}$	The number of time intervals on the horizon
$\boldsymbol{\nu}$	A 30 day time interval
$\boldsymbol{t}$	Time interval
$\boldsymbol{\vec{d_r}}$	Vector of demand for reserved instances over τ
$\boldsymbol{\mu_r}$	Average of on reserved customers at each interval
$\boldsymbol{\vec{u}}$	Utilization over the τ
$\boldsymbol{\mu_u}$	Average of utilization for reserved instances at each interval
$\boldsymbol{\sigma_u}$	Standard deviation of utilization for reserved instances at each interval
$\boldsymbol{\vec{d_o}}$	Vector of demand for on-demand instances over τ
$\boldsymbol{\mu_o}$	Average of on on-demand customers at each interval

$\vec{d_s}$	Vector of demand for spot instances over τ
μ_s	Average of on spot customers at each interval
p	Price of on-demand instances
φ	Upfront fee for reserved instances
α	Discount factor for reserved instances daily usage fee
β	Discount factor for on-demand users from the queue
p_s	Price of spot instances in model 1.0
Δ	Constant threshold of spot instances
Δ_t	Threshold at each interval
r_t	Number of reserved users accepted to the cloud at time t
l_t^r	Number of reserved users in the cloud at time t
o_t	Number of on-demand users accepted to the cloud at time t
l_t^o	Number of on-demand users in the cloud at time t
q_t	Number of on-demand users accepted to the cloud from queue at time t
l_t^q	Number of on-demand users in the cloud from queue at time t
$\vec{Q}_t$	Set of on-demand users in the queue
s_t	Number of spot users accepted to the cloud at time t
l_t^s	Number of spot users in the cloud at time t
$\vec{\pi_t}$	Set of all bids at time t
μ_π	Average value of the bids
σ_π	Standard deviation of the bids
Π_t	Set of spot bids with their service duration

ϑ_t	Set of live spot bids, their service duration, and expected revenue
ω_j	Service duration of jth on-demand user
χ_j	Leaving time of jth on-demand user
κ_t	Number of on-demand users leaving at time t
ζ_t	Revenue of the cloud for period t
Z	Total revenue of the cloud

Appendix B

Python Code of Model 3.1

As model 3.1 has the most elaborate algorithm we have provided the python code of this model for clarification purposes.

```
import numpy as np
import math
import csv
from operator import itemgetter



def revenue_cal(
        trial, intervals, month_hour, capacity, rho, price,
        phi, alpha, on_demand_life_ave, spot_life_ave
):

    """
    defining the parameters:
    """
        # reservation discount factor
    def psi_t(x): return (intervals - x) / month_hour
```

```
    """
    generating the demands
    """
    # reading the demand matrix
    filename = "data" + str(trial)
    reader = csv.reader(open('%s.csv' % filename, "rt"),
        delimiter=",")
    temp = list(reader)
    data = np.array(temp).astype("float")

    # storing demands of on-demand
    demand_r = (np.array(data)[:, 0]).astype("int")

    # storing demands of on-demand
    demand_o = (np.array(data)[:, 1]).astype("int")

    # storing demands of on-demand
    demand_s = (np.array(data)[:, 2]).astype("int")

    # storing the utilization at each period
    utilization = np.array(data)[:, 3]

    """
    Building the simulation and calculating
        the revenue at each time window
    """
    dec_r = np.zeros(shape=intervals, dtype=int)
    dec_o = np.zeros(shape=intervals, dtype=int)
    dec_s = np.zeros(shape=intervals, dtype=int)
    s_r = np.zeros(shape=intervals, dtype=int)
    s_rl = np.zeros(shape=intervals, dtype=int)
```

```
s_o = np.zeros(shape=intervals, dtype=int)
s_s = np.zeros(shape=intervals, dtype=int)
s_c = np.zeros(shape=intervals, dtype=int)
rev_r = np.zeros(shape=intervals)
rev_o = np.zeros(shape=intervals)
rev_s = np.zeros(shape=intervals)
rev_t = np.zeros(shape=intervals)
threshold_s = np.zeros(shape=intervals)
l_rt = 0
l_ot = 0
l_qt = 0
h_j = np.zeros(shape=intervals)
g_j = np.zeros(shape=intervals)
live_spot = []
threshold = 0.3144
spot_pool = []

# queueing system parameters:
q_number = np.zeros(shape=14, dtype=int)

for i in range(intervals):
    """
    finding the decisions
    """
    l_st = np.size(live_spot, 0)
    r_t = min(capacity - (l_rt + l_ot + l_qt),
              math.floor(demand_r[i] * rho))
    o_t = int(min(capacity - (
              r_t + l_rt + l_ot + l_qt), demand_o[i]))
    q_potential = demand_o[i] - o_t
```

```
# we check the queue to see...
#if we can add more on_demand
q_t = 0
empty_cap = (capacity - (r_t + l_rt + o_t +
        l_ot + l_qt))
if empty_cap > 0 and sum(q_number) > 0:
    flag = True
    indice = 13
    while flag and indice > -1:
        if (q_number[indice] - empty_cap) > 0:
            q_t += empty_cap
            q_number[indice] -= empty_cap
            flag = False
        else:
            q_t += q_number[indice]
            empty_cap -= q_number[indice]
            q_number[indice] = 0
            indice -= 1
q_t = int(q_t)

# updating the number of instances in the queue
for b in range(13):
    index = b + 1
    q_number[-index] = q_number[-(index+1)]
q_number[0] = int(q_potential*np.random.uniform(low=0.3,
        high=0.8))
```

```
# calculating the number of spot instances...
# that will pass the threshold:
# the array of the bids of spot instances and...
# updating the array of live spot instances
mu, sigma = threshold + 0.024, 0.03
shape, scale = (mu/sigma)**2, (sigma**2/mu)
spot_price = np.random.gamma(shape=shape,
        scale=scale, size=demand_s[i])
# substituting those prices less than 0.1 with 0.1
spot_price[spot_price < 0.1] = 0.1
for u in range(demand_s[i]):
# the lifetime of spot are half the on-demand
    temp2 = np.random.geometric(p=spot_life_ave)
    spot_pool.append([spot_price[u], temp2,
                spot_price[u]*temp2])

# sorting the spot pool decreasingly (based on the bid)
spot_pool = sorted(spot_pool, key=itemgetter(0),
        reverse=True)

if (capacity - ((math.floor((r_t + l_rt)
        * utilization[i])) + o_t + l_ot + q_t
        + l_qt + l_st)) > 0:
    s_t = int(min(capacity - (math.floor((r_t + l_rt)
                * utilization[i]) + o_t + l_ot + q_t
                + l_qt + l_st), demand_s[i]))
else:
    s_t = 0
```

```
# fill the available spot instances into the system
spot_number = int(capacity - ((math.floor((r_t + l_rt)
        * utilization[i])) + o_t + l_ot + q_t + l_qt))
if spot_number < np.size(spot_pool, axis=0):
    live_spot = spot_pool[:spot_number]
else:
    live_spot = spot_pool

# number of live spot instances in the system
l_st = np.size(live_spot, 0)


dec_r[i] = r_t
dec_o[i] = o_t + q_t
dec_s[i] = s_t

"""
calculating the revenue at each time window
"""
if i < (intervals - month_hour):
    rev_r[i] = r_t * phi + alpha * price
                * math.floor((r_t + l_rt)
                * utilization[i])
else:
    rev_r[i] = psi_t(i) * r_t * phi + alpha
                * price * math.floor((r_t + l_rt)
                * utilization[i])


rev_o[i] = (price * (o_t + l_ot)) + (0.8 * price
        * (q_t + l_qt))
rev_s[i] = np.sum(live_spot, axis=0)[0]
rev_t[i] = (rev_r[i] + rev_o[i] + rev_s[i])
```

```
# updating l_rt, l_ot and l_st
l_rt = l_rt + r_t

# the life time of on-demand instances
for w in range(o_t):
    temp1 = np.random.geometric(p=on_demand_life_ave)
    if (i + temp1 + 2) < intervals:
        h_j[i + temp1] += 1

# the life time of on-demand instances from the queue
for w in range(q_t):
    temp2 = np.random.geometric(p=on_demand_life_ave)
    if (i + temp2 + 2) < intervals:
        g_j[i + temp2] += 1

# updating live on-demand instances
l_ot += o_t
l_qt += q_t

# updating live spot instances regarding
# the leaving time
for q in range(np.size(live_spot, 0)):
    live_spot[q][1] -= 1
    live_spot[q][2] = live_spot[q][0]
                * live_spot[q][1]
live_spot = [z for z in live_spot if z[1] > 0]

# spot pool now only contains live spot instances
spot_pool = live_spot
```

```
        # the capacity left in the system is
        left_capacity = capacity - (math.floor(l_rt *
                utilization[i])
                + l_ot + l_qt + l_st)

        # state of the system
        s_r[i] = l_rt
        s_rl[i] = math.floor(l_rt * utilization[i])
        s_o[i] = l_ot + l_qt
        s_s[i] = l_st
        s_c[i] = left_capacity

        # threshold of spot instances
        threshold = live_spot[-1][0]
        if threshold > 0.8:
            threshold = 0.8
        threshold_s[i] = threshold

        # updating the live reserved, on-demands:
        l_ot = l_ot - h_j[i]
        l_qt = l_qt - g_j[i]
        if i >= month_hour:
            l_rt = l_rt - dec_r[i - month_hour]

    return demand_r, demand_o, demand_s, dec_r, dec_o,
        dec_s, s_r, s_rl, s_o, s_s, s_c, rev_r, rev_o,
        rev_s, rev_t, threshold_s
```

Bibliography

[1] Thoughts on cloud, June 2017. URL https://www.ibm.com/blogs/cloud-computing/.

[2] Guofu Feng, Saurabh Garg, Rajkumar Buyya, and Wenzhong Li. Revenue maximization using adaptive resource provisioning in cloud computing environments. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 192–200. IEEE Computer Society, 2012.

[3] Jeffrey I McGill and Garrett J Van Ryzin. Revenue management: Research overview and prospects. *Transportation science*, 33(2):233–256, 1999.

[4] Hong Xu and Baochun Li. Dynamic cloud pricing for revenue maximization. *IEEE Transactions on Cloud Computing*, 1(2):158–171, 2013.

[5] What is data center? - definition from whatis.com. URL http://searchdatacenter.techtarget.com/definition/data-center.

[6] 1940s — timeline of computer history. URL http://www.computerhistory.org/timeline/1941/.

[7] Eamonn Colman. Comparison of saas, paas, and iaas. URL https://www.computenext.com/blog/when-to-use-saas-paas-and-iaas/.

[8] Amazon web services, inc. URL https://aws.amazon.com/documentation/.

[9] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.

[10] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Computing Surveys (CSUR)*, 47(4):63, 2015.

[11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[12] Wendell R Smith. Product differentiation and market segmentation as alternative marketing strategies. *Journal of marketing*, 21(1):3–8, 1956.

[13] Wei Wang, Baochun Li, and Ben Liang. Towards optimal capacity segmentation with hybrid cloud pricing. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 425–434. IEEE, 2012.

[14] Leonard Heilig and Stefan Voß. Decision analytics for cloud computing: a classification and literature review. *Tutorials in Operations Research–Bridging Data and Decisions*, pages 1–26, 2014.

[15] May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud computing pricing models: a survey. *International Journal of Grid & Distributed Computing*, 6(5):93–106, 2013.

[16] Cuong T Do, Nguyen H Tran, Eui-Nam Huh, Choong Seon Hong, Dusit Niyato, and Zhu Han. Dynamics of service selection and provider pricing game in heterogeneous cloud market. *Journal of Network and Computer Applications*, 69:152–165, 2016.

[17] Bhanu Sharma, Ruppa K Thulasiram, Parimala Thulasiraman, and Rajkumar Buyya. Clabacus: a risk-adjusted cloud resources pricing model using financial option theory. *IEEE Transactions on Cloud Computing*, 3(3):332–344, 2015.

[18] Shifeng Shang, Jinlei Jiang, Yongwei Wu, Zhenchun Huang, Guangwen Yang, and Weimin Zheng. Dabgpm: A double auction bayesian game-based pricing

model in cloud market. In *IFIP International Conference on Network and Parallel Computing*, pages 155–164. Springer, 2010.

[19] Wubin Li, Petter Svärd, Johan Tordsson, and Erik Elmroth. Cost-optimal cloud service placement under dynamic pricing schemes. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 187–194. IEEE Computer Society, 2013.

[20] Adel Nadjaran Toosi, Farzad Khodadadi, and Rajkumar Buyya. Sipaas: Spot instance pricing as a service framework and its implementation in openstack. *Concurrency and Computation: Practice and Experience*, 2015.

[21] Fadi Alzhouri and Anjali Agarwal. Dynamic pricing scheme: Towards cloud revenue maximization. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 168–173. IEEE, 2015.

[22] Hai Jin, Xinhou Wang, Song Wu, Sheng Di, and Xuanhua Shi. Towards optimized fine-grained pricing of iaas cloud platform. *IEEE Transactions on cloud Computing*, 3(4):436–448, 2015.

[23] Wei-Yu Lin, Guan-Yu Lin, and Hung-Yu Wei. Dynamic auction mechanism for cloud resource allocation. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 591–592. IEEE, 2010.

[24] Adel Nadjaran Toosi, Kurt Vanmechelen, Kotagiri Ramamohanarao, and Rajkumar Buyya. Revenue maximization with optimal capacity control in infrastructure as a service cloud markets. *IEEE Transactions on Cloud Computing*, 3(3):261–274, 2015.

[25] Pricing, Amazon Web Services, Inc. https://aws.amazon.com/ec2/pricing/. Accessed: 2017.

[26] Hai Xie, Sami Tabbane, and David J Goodman. Dynamic location area management and performance analysis. In *Vehicular Technology Conference, 1993., 43rd IEEE*, pages 536–539. IEEE, 1993.

[27] Henk C Tijms. *A first course in stochastic models*. John Wiley and sons, 2003.

[28] Baris Selçuk and Özgür Özlük. Optimal keyword bidding in search-based advertising with target exposure levels. *European Journal of Operational Research*, 226(1):163–172, 2013.