CAPACITY ALLOCATION AND PRICING POLICIES FOR CLOUD COMPUTING SERVICE PROVIDERS

R. Afghah[1] and Ö. Özlük[2] and T. Ünlüyurt[3*]
[1]Smith School of Business
Queens University, Canada
17smra@queensu.ca

[2]Department of Industrial Engineering
MEF University, Istanbul, Turkey
ozluko@mef.edu.tr

[2]Faculty of Engineering and Natural Sciences, Industrial Engineering Program
Sabanci University, Istanbul, Turkey
tonguc@sabanciuniv.edu

ABSTRACT

The cloud computing is regarded as a paradigm shift in today's IT world. As cloud computing resources behave like perishable products, revenue management techniques can be applied to increase cloud service provider's total revenue. In this paper, we propose various methods for pricing and capacity allocation. We consider three types of instances offered by the service provider; subscription, on-demand and spot instances. We introduce three allocation and pricing policies and propose different models. We simulate these models on a randomly generated dataset and evaluate the models for different capacities. The results we obtain indicate the sensitivity of revenue to varying policies and demonstrate the potential profit increase for cloud service providers.

Keywords: Cloud computing, pricing, simulation

1 INTRODUCTION

Since Amazon introduced its Elastic Compute Cloud in 2006, cloud computing has become a fundamental way of using the internet to access computing resources as services. In a report by Forrester Research, the global cloud market is expected to reach $241 billion in 2020. Wikipedia defines cloud computing as "a type of Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand." This revolutionary technology provides a venue for scalable and –almost– on-demand sharing of resources and setup/operating expenses among a large number of end users. Using cloud computing, end users can collect, store, manage, process and analyze data in an efficient and affordable manner at high computing speeds. In most cases, users of cloud systems are not required to install any software and are able to reach their data from a computer with an Internet connection regardless of where they are. As a result of all these, cloud computing has also been an important catalyzer in the spread and democratization of technology entrepreneurship.

Cloud computing service provider companies maintain large scale data-centers and provide multiple categories of services. There are three main services offered by these companies: Software as a Service(SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). SaaS is the most familiar configuration of cloud services for customers, it transfers the function of software administration and its installation to third-party services. Examples for the application this type of service are comprehensive such as Salesforce, productivity software like Google Apps, and storage solutions like Dropbox. PaaS operates at a lower tier of the data center than SaaS, typically offering software developers a platform, where they can develop and deploy their software. 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. Heroku, Red Hat™s OpenShift, and Google App Engine are examples of PaaS providers. 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. 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. The three main service providers in this area are Azure, Amazon EC2, and Google Cloud Platform.

The focus of this study is the pricing models used 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. To compare the different pricing models used by EC2, we require to understand the concept of Service Level Agreement (SLA). SLA is a contract between a service provider and its internal/external customers that documents what services the provider will furnish and defines the performance standards the provider is obligated to meet. Some metrics offered by Amazon that SLAs may specify include: the proportion of time which services will be available (availability/uptime), application response time, notification of network changes in advance that might affect user's performance. An SLA specifies availability, performance, memory storage and other parameters for distinct classes of service infrastructure. Based on the commitments promised by the SLAs, Amazon has three dominant pricing plans for its IaaS:

Subscription-based services (reserved service) allow the user or business organization to purchase (subscribe to) 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. 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 the subscription, the longer its term the less its price per time unit will become.

On-demand services let the end-user purchase a computing instance over a random amount of time for a specific service which they requested; in this case, a consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

Spot Instances enable a user to bid on unused capacity of the service provider. 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. A user's spot instance runs whenever his/her bid exceeds the current market price.

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. The goal of this study is to investigate how a service cloud provider with limited computing capacity can optimally allocate its resources to the three types of services mentioned above. Furthermore, we will examine the effect of several pricing policies in total revenue of a service provider.

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 [6]. In addition, the contribution of this paper is that 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 a 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 at a discounted price.

In Section 2, we discuss the previous work that has been done on the subject of data center's revenue improvement and cloud computing on a broader sense. In Section 3, we will present the models and pricing policies that have been analyzed. Section 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, we present our conclusions and provide directions for future work.

2 LITERATURE SURVEY

There has been a vast literature in cloud computing regarding the pricing of cloud services. In 2016, Do et al [1], they 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; their model extracts the equilibrium prices of the non-cooperative static game and successively the dynamic of the cloud users in the selection game.

Sharma et al [2] 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.

In 2010, Shang et al [3] work with auction based pricing policy for cloud computing services, using a double auction Bayesian game based algorithm. One major restriction in their model is that they consider the price of the providers and the users to be uniformly distributed.

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 [4] 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.

Similarly, Jin et al [5] introduces 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 virtual machine 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.

Toosi et al [6] 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. However, their model lacks authenticity due to its oversimplification estimating the reservation utilization and the pricing fluctuation of spot instances.

Toosi [6] 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). 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.

3 SYSTEM MODEL AND METHODOLOGY

This section 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 section. First, 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, which indicates the prices of m4.large offered by Amazon.

Table 1: m4.large Pricing Plans

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

3.1 Cloud Pricing Plans

The model considers three pricing plans that are decribed next in detail.

3.1.1 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.

3.1.2 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 is 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 $\varphi + \alpha\, p\, t$.

3.1.3 Spot Instances

In this plan, the customers will bid on instances in an auction-based market. If their bid is higher than the threshold of the IaaS provider, then they are granted the access to the instance. However, this plan has the lowest QoS (Quality of Service). 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 [7]. Spot instance's price is regulated by a discounted rate $p_s = \$0.3144$, and the price at each billing cycle is quantified by their bids.

3.2 Proposed Model

The model that we propose simulates the cloud system and examines the dynamics of the system and the revenue of the provider for different pricing 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. The demands are generated acoording to a Poisson distribution with parameters 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 framework 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 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 and σ. 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 incoming demand of on-demand instances, we accept as many on-demand instances as possible. Whereas reserved instances may sometimes 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 instance life time is 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 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 general framework that is just described, we can present the general assumptions used in this study in the list below:

- We run each scenario on two different capacities; first, the case where we have capacity shortage (1500). We also evaluate the revenues on excessive scarcity of resources (1000). The average capacity is calculated by the summation of three arguments; (i) product 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.

$$max\_average\_capacity = \ m_u\ (30\ \mu_r) + 11\ \mu_o + 5.5\ \mu_s$$

- We assume that the rejected customers or the ones that leave the system will not reenter the cloud.
- 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 ρ 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 μ_ω = 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

We have devised 3 models to test their revenue over the time horizon of 360 days. These algorithms are based on the framework used by Toosi *et al*. in 2015. We start by Toosi's model and move on to more realistic models that will increase the revenue.

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.

The first step of the model is to find the number of reserved instances that can be received by the system. The number of reserved instances accepted to the cloud at period t is denoted by 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. So, the value of the r_t can be computed by equation (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.

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

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_t^r requires to be updated after each iteration according to:

$$l_t^r = l_{t-1}^r + r_t, \qquad \forall t \in (1,\nu) \tag{2}$$

$$l_t^r = l_{t-1}^r + r_t - r_{t-\nu}, \qquad \forall t \in (\nu,\tau) \tag{3}$$

This is based on the fact that the instances will leave the system after they finish their 30 days (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.

$$o_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o,\ d_t^o), \forall t \in (1,\tau) \tag{4}$$

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 is subtracted from the number of live on-demand instances, i.e. $l_t^o = l_{t-1}^o + o_t - \kappa_t$, 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.

$$\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\kappa_t = \sum_j \chi_j(t) \quad \forall t \in (1, \tau) \tag{6}$$

$$l_t^o = l_{t-1}^o + o_t - \kappa_t \quad \forall t \in (1, \tau) \tag{7}$$

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. Those instances that 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 [6]. 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.

$$s_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o,\ d_t^s), \qquad \forall t \in (1, \tau) \tag{8}$$

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 (9) 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 (10)). 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.

$$\zeta_t = \varphi r_t + \alpha p u_t(r_t + l_t^r) + p(o_t + l_t^o) + p_s\, s_t, \qquad \forall t \in (1, \tau - \nu) \tag{9}$$

$$\zeta_t = \psi(t)\varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) + p_s\, s_t, \forall t \in (\tau - \nu + 1, \tau) \tag{10}$$

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.

$$Z = \sum_{t=1}^{\tau} \zeta_t, \qquad \forall t \in (1, \tau) \tag{11}$$

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 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. 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 ∈ $(1,\tau)$). The generation of each bid is done by Gamma distribution. We have used Gamma distribution due to its property of being a very flexible non-negative random generator [8]. This distribution has been used for the generation of bids by a random variable in literature, such as the study by Selçuk and Özlük [9]. 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 $\mu_\pi = \Delta + \varepsilon$. The vector $\vec{\pi}_t$ all the bids of the customer at time t, and it is derived as the following:

$$\pi_t^i = random.gamma(\mu_\pi, \sigma_\pi), \qquad \forall i \in (1, d_t^s) \tag{12}$$

$$\vec{\pi}_t = \{\pi_t^i | \quad \forall i \in (1, d_t^s)\} \tag{13}$$

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:

$$\Pi_t = \{(\pi_t^i, \eta^i) | \quad \forall i \in (1, d_t^s)\} \tag{14}$$

Then, the algorithm will check the empty capacity left in the cloud regardless of the live spot instances in the system, for potential users. The service provider does 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.

$$\varpi_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o,\ d_t^s), \qquad \forall t \in (1,\tau) \tag{15}$$

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_t^s.

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:

$$\vartheta_t(m,2) = \vartheta_t(m,2) - 1, \qquad \forall t \in (1,\tau) \tag{16}$$

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:

$$\zeta_t^s = \sum_m \vartheta_t(m,1), \qquad \forall t \in (1,\tau) \tag{17}$$

By having ζ_t^s 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 calculated 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 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. $\{4,6,3,8,\dots,7\}$ 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 users 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:

$$q_t = min(C - r_t - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ \sum_j Q_t), \forall t \in (1,\tau) \tag{18}$$

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:

$$\chi_j(t) = \begin{cases} 1 & \text{if} \quad t = e_j \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$\kappa_t = \sum_j \chi_j(t) \quad \forall t \in (1,\tau) \tag{20}$$

$$l_t^q = l_{t-1}^q + q_t - \kappa_t \quad \forall t \in (1,\tau) \tag{21}$$

By having these variables introduced to our system, th equations will be updated respectively to equations (22), (23), and (24):

$$r_t = min(C - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ \rho\, d_t^r), \qquad \forall t \in (1,\tau) \tag{22}$$

$$o_t = min\left(C - r_t - l_{t-1}^r - l_{t-1}^o - l_{t-1}^q,\ d_t^o\right), \quad \forall t \in (1,\tau) \tag{23}$$

$$\varpi_t = min(C - u_t(r_t + l_{t-1}^r) - l_{t-1}^o - l_{t-1}^q,\ d_t^s), \qquad \forall t \in (1,\tau) \tag{24}$$

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

$$\begin{aligned} \zeta_t = \quad & \varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) \\ & \beta p(q_t + l_{t-1}^q) + \sum_m \vartheta_t(m,1), \qquad \forall t \in (1,\tau - \nu) \end{aligned} \tag{25}$$

$$\begin{aligned} \zeta_t = \quad & \psi(t)\varphi r_t + \alpha p u_t(r_t + l_{t-1}^r) + p(o_t + l_{t-1}^o) \\ & \beta p(q_t + l_{t-1}^q) + \sum_m \vartheta_t(m,1), \qquad \forall t \in (\tau - \nu + 1,\tau) \end{aligned} \tag{26}$$

4 DATASET AND RESULTS

In this section, we introduce the corresponding computational environment for the simulations, and then move on to discussing the dataset used for this study. Moreover, we will present the results of each simulation runs and analyze the total revenue produced by each model. Finally, we argue about the findings and interpret the results. The simulation models were executed on a host computer with a Debian-based Linux operating platform (Linux 16.04 LTS) with a Dual-Core Intel(R) Core(TM) i5-5200U Processor. The models presented in the previous section were coded in python programming language and we used python 3.5 as its interpreter.

4.1 Dataset

As revenue management is a competitive subject amongst CSPs they do not disclose their demands to the 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 1000, and 1500 to investigate the effect of capacity on determining the optimal pricing policy.

In our dataset, the customer arrivals are based on a constant rate according to a Poisson distribution. The parameters of this dataset will be:

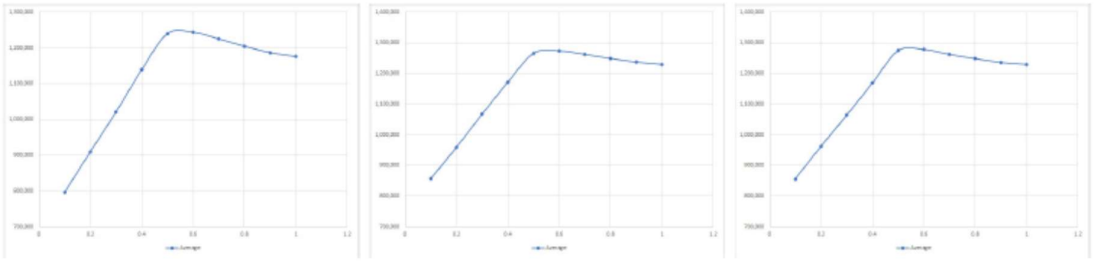
Table 2: Parameters of Demand Generation

Demand Type	*Regular*
μ_r	50
μ_o	60
μ_s	300
μ_u	0.50
σ_u	0.15

4.2 Results

The output of the simulation algorithms are presented in this section, we categorized them by 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.

Capacity 1500



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

Figure 1: Total Revenue for Capacity 1500

Initially, we inspect every model's revenue according to the change in ρ. Figure 1 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 afterward. 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. We observe that Models 2.0 and 2.1 generates more revenue compared with Model 1.0.

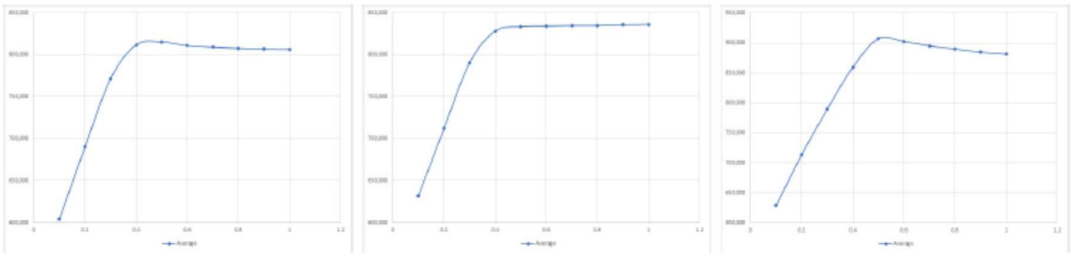
Mainly, for the proposed models a ρ equal to 0.6 yields the highest revenue in this dataset. we have provided the list maximum revenues in table 9 for all pricing models.

Table 3: Maximum Revenues for Capacity 1500

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1500	0.6	1239096	1243509	1247922
Model 2.0	1500	0.6	1269961	1273489	1277018
Model 2.1	1500	0.6	1274823	1278556	1282289

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 2).



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

Figure 2: Total Revenue for Capacity 1000

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. 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. 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 11. Similar to the case with capacity 1000, we observe higher revenues with Models 2.0 and 2.1. In this case, the percent increases in the revenues are even higher.

Table 4: Maximum Revenues for Capacity 1000

Model Version	*Capacity*	ρ	Lower Bound	Average	Upper Bound
Model 1.0	1000	0.3	871843	873633	875423
Model 2.0	1000	0.2	878864	880488	882111
Model 2.1	1000	0.2	888741	892396	896051

5 CONCLUSION

In this paper, we propose different models for determining capacity allocation and pricing for cloud computing service providers. We evaluate each model by simulation and observe that it is possible to increase revenue by making better decisions regarding capacity allocation and pricing. As a natural continuation of this work, one can develop more realistic models by considering the dynamics of cloud computing environment and evaluate the models by using different datasets that are more realistic.

6 REFERENCES

[1] **Do, C.T., Tran, N.H., Huh, E.N., Hong, C.S., Niyato, D. and Han, Z.** 2016. Dynamics of service selection and provider pricing game in heterogeneous cloud market. Journal of Network and Computer Applications, 69, pp.152-165.

[2] **Sharma, B., Thulasiram, R.K., Thulasiraman, P. and Buyya, R.** 2015. Clabacus: a risk-adjusted cloud resources pricing model using financial option theory. IEEE Transactions on Cloud Computing, 3(3), pp.332-344.

[3] **Shang, S., Jiang, J., Wu, Y., Huang, Z., Yang, G. and Zheng, W.** 2010. DABGPM: A double auction Bayesian game-based pricing model in cloud market. In IFIP international conference on network and parallel computing (pp. 155-164).

[4] **Xu, H. and Li, B.** 2013. Dynamic cloud pricing for revenue maximization. IEEE Transactions on Cloud Computing, 1(2), pp.158-171.

[5] **Jin, H., Wang, X., Wu, S., Di, S. and Shi, X.** 2015. Towards optimized fine-grained pricing of IaaS cloud platform. IEEE Transactions on cloud Computing, 3(4), pp.436-448.

[6] **Toosi, A.N., Vanmechelen, K., Ramamohanarao, K. and Buyya, R.** 2015. Revenue maximization with optimal capacity control in infrastructure as a service cloud markets. IEEE transactions on Cloud Computing, 3(3), pp.261-274.

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

[8] **Tijms, H.C.** 2003 A first course in stochastic models. John Wiley and Sons.

[9] **Selcuk, B. and Ozluk, O.** 2013. Optimal keyword bidding in search-based advertising with target exposure levels. European Journal of Operational Research, 226(1):163-172.