MODELING PRICE AND DEMAND IN CLOUD COMPUTING SYSTEMS

BY

KEVIN DOHERTY SCHMID

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Bruce Hajek
ABSTRACT

The purpose of this thesis is to explore the behavior of pricing and demand within cloud computing systems. As cloud computing has become increasingly popular, research has been focused on finding a mechanism to optimize social welfare. Current schemes typically charge users at a fixed price per time unit. Theoretical research alternatives to this include combinatorial auctions. Analyses of these systems often utilize a simplistic model of a user. This thesis aims to introduce a more realistic user model and analyze optimization from the user’s perspective in an environment similar to that of the Amazon EC2 spot instance market. It is shown that the optimal strategy of a user with a deadline and sequential load can be derived from a set of dynamic programming equations, the results of which can be implemented without knowledge of the current price. Additionally, it is shown that this strategy is closely approximated by an expected rate user model. Finally, it is shown that these types of users are elastic and the load on the system can be controlled through the price mechanism.
I would like to acknowledge and thank my advisor, Professor Hajek. His help in finding a suitable research topic and weekly discussions on the material were invaluable towards the completion of this thesis. I would also like to acknowledge my roommate, Diego Hernandez, whose constant support was greatly appreciated. Finally, I would like to thank my family, whose efforts in the academic fields have always been a source of inspiration to me.
TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION ............................................. 1
  1.1 Cloud Computing .............................................. 1
  1.2 Overview .................................................... 2

CHAPTER 2 LITERATURE REVIEW ................................. 3

CHAPTER 3 MODEL DEVELOPMENT ................................. 5
  3.1 Price Model ................................................... 5
  3.2 Sequential Load Users ....................................... 8
  3.3 The Impact of Price Stickiness .............................. 16
  3.4 User Elasticity .............................................. 20

CHAPTER 4 SIMULATION AND RESULTS ........................... 22
  4.1 Inelastic Load ............................................... 22
  4.2 Elastic Load ................................................. 26
  4.3 Load Controlled Auction .................................... 28

CHAPTER 5 CONCLUSION AND FUTURE WORK .................... 33

REFERENCES ......................................................... 35

APPENDIX A MONOTONICITY OF OPTIMAL CONTROL ............. 36
1.1 Cloud Computing

The emergence of the Internet has led to the development of large cloud computing centers by companies including Microsoft, Amazon, and Google. Cloud computing allows providers to profit from unused hardware by selling virtual machines on the cloud. Individual users and businesses can purchase computational power without having to pay the overhead cost of hosting the hardware [1].

One of the cloud computing providers that has received significant attention from the research community is Amazon. Typically, cloud providers sell virtual machines at a fixed cost; however, Amazon provides the following three options:

- Dedicated on-demand instances can be bought at a fixed price; the user pays for however long it uses the virtual machines.

- Reserved instances can be bought at a fixed price; the user pays for one to three year contracts. These come at rates heavily discounted from the on-demand price.

- Spot instances can be bid on and bought at a variable price.

This last option provides an auction based mechanism for the sale of virtual machines and is the option focused on in this paper.

The Amazon EC2 spot instance market offers a wide variety of virtual machines ranging in processing power, memory, and disk space. To enter the market, a user simply specifies its computational needs and a bid which represents the most it would be willing to pay for the service. The current spot price is set by Amazon and is a function of demand. It is unknown to the users, but a history of previous prices is provided. If the user’s bid is higher
than the current price, the user will receive service and pay at the current price. A spot price can change at any point in time and if a user’s bid dips below it, service is suspended. In this market, there is no guarantee of continuous service so it is only useful for users with time-flexible, interruption tolerant tasks. If used appropriately, it can provide service at a significantly discounted rate of the on-demand price [2].

1.2 Overview

The goal of this thesis is to provide a simple but realistic cloud user model that can be used in further analysis and optimization of cloud computing environments. In order to achieve this, the thesis is broken into several parts. Chapter 2 gives a description of current research focusing on the development of mechanisms which can increase the efficiency of cloud computing exchanges. Chapter 3 develops a pricing model based on the Amazon EC2 spot instance market and a model for users called a sequential load user model. Two types of sequential load users are introduced, the dynamic program (DP) sequential load user and the expected rate (ER) sequential load user. Special consideration is given to how certain parameters affect the user’s behavior within each of the models. Chapter 4 provides the results of the models in an auction simulation and introduces a mechanism to control the user load. Chapter 5 describes how the results of this thesis could be used for future work within the field of cloud computing.
Cloud computing is still a relatively new technology and there are many opportunities for research in the interactions between the providers and users. It was not until 2009 that Amazon started offering an auction based market for its cloud computing services [3]. One of the more impactful areas of research has been the development of mechanisms that yield a more economically efficient environment. Zaman and Grosu [4] claim that the fixed price model cannot allocate virtual machine instances efficiently and that a combinatorial auction would be particularly well suited to solving the resource allocation problem [4]. Work by Samimi, Teimouri, and Mukhtar [5] argues that a more efficient mechanism would be a combinatorial double auction in which brokers and auctioneers act as middle men between the users and providers. Unfortunately, the combinatorial auction resource allocation problem is NP-hard. In continued work, Zaman and Grosu [6] offer two algorithms for approximate solutions and show that improvements can be made over the fixed price model at the cost of algorithm efficiency.

In a similar area Shi, Zhang, Wu, Li, and Lau [7] analyze the competitive ratio of an auction mechanism where users can bundle different virtual machine types. They give a solution to the offline social welfare optimization problem and provide an algorithm to solve the online social welfare optimization problem. Furthermore, they show the mechanism is truthful, computationally efficient, and approaches a competitive ratio of \( e + \frac{1}{e-1} \).

Yet another area of research is to analyze the market mechanisms that currently exist. Most work in this field has been centered on the Amazon EC2 spot instance market. The algorithms behind Amazon’s determination of the spot instance prices are not publicly known. Research has used the history of spot prices to try to reverse engineer the mechanisms used and understand the effect prices have on demand. Wee [8] claims that spot instances are 52% cheaper on average than the on-demand price but that the price fluctuations have not been meaningfully changed over time to shift demand to off-peak hours. Ben-Yehuda, Schuster,
and Tsafrir [3] argue that the spot prices are typically generated at random from a tight interval with a dynamic reserve price.

Finally, an additional area of research is the optimization of these cloud computing systems from a user’s perspective. Research in this area has come mainly as a side effect of the other areas. To show efficiency of algorithms or improvements in economic value, researchers have created simplistic models to approximate the user behavior. There is a need for tractable and realistic user models, which is where this thesis draws its inspiration.
CHAPTER 3
MODEL DEVELOPMENT

The Amazon EC2 Spot Instance market provides an auction based mechanism for users to bid on virtual machines. A user specifies its computational demands and bid prices. If the current spot price is less than a user’s bid, the user receives service and pays the spot price. This can be considered a second price auction in which the user’s dominant strategy is to bid its true valuation of the service [9]. The spot price is set by Amazon at any given time and is supposed to change as a result of demand. The models that are developed and analyzed in this chapter assume that users are both truthful and rational.

3.1 Price Model

In order to analyze the behavior of a user in an auction based marketplace similar to that of the Amazon EC2 Spot Instance market, some of its features are simplified. In the following pricing model we assume a single virtual machine type and a constant hourly change of spot prices. Furthermore we assume that the spot prices take values in a finite set and that their distribution is known. Let $d$ be a positive integer that represents the number of pricing states in the model. Then $q = (q_i : i \in \{0, 1, ..., d - 1\})$ represents the distribution of price states and $\gamma = (\gamma_i : i \in \{0, 1, ..., d - 1\})$ represents the spot price as a percentage of the on-demand price in each of the states. We assume that $\gamma_i$ is always greater than zero and less than one, implying that the spot price is always less than the on-demand price. This almost always holds in the Amazon EC2 spot market and rarely does the spot price exceed the on-demand price [3]. Without loss of generality the pricing states can be arranged in order of increasing price such that $0 < \gamma_0 < \gamma_1 < ... < \gamma_{d-1} < 1$. Finally, we introduce a stickiness parameter $\theta$ which represents the probability the market stays at its current price from one hour to the next.
In this model the spot price is given by a Markov chain with the state space \( \{0, 1, \ldots, d-1\} \). Given the current state \( i \), the next state \( j \) is chosen according to the following probability:

\[
p_{i,j} = \theta \delta_{i,j} + (1 - \theta)q_j
\]

where \( \delta_{i,j} = 1 \) if \( i = j \) and \( \delta_{i,j} = 0 \) if \( i \neq j \). A value of \( \theta = 1 \) corresponds to a fixed price model, in which the price is constant over time. A value of \( \theta = 0 \) corresponds to a model with independent prices from one hour to the next, each following the distribution of \( q \). Based on the spot price histories given by Ben-Yehuda [3] and Grosskur [10] we see that the prices have essentially bimodal distributions with high probabilities in low and high states, and low probabilities in the middle states. In order to model this in simulation, the parameters given in Table 3.1 were used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d )</td>
<td>7</td>
<td>number of price states</td>
</tr>
<tr>
<td>( q )</td>
<td>([0.05, 0.3, 0.2, 0.05, 0.1, 0.2, 0.1])</td>
<td>price state distribution</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>([0.2, 0.25, 0.33, 0.5, 0.66, 0.75, 0.9])</td>
<td>prices as on-demand percentage</td>
</tr>
<tr>
<td>( \theta )</td>
<td>([0,1])</td>
<td>price stickiness</td>
</tr>
</tbody>
</table>

To give an impression of the significance of the stickiness parameter \( \theta \), Figures 3.1 and 3.2 show the states of the price model simulated over a week, \( W = 168 \) hours. In Figure 3.1, \( \theta = 0 \) and the price jumps significantly from one hour to the next according to the distribution \( q \). In Figure 3.2, \( \theta = .85 \) the price changes at a much less frequent pace.
Figure 3.1: Price states over time for $\theta = 0$

Figure 3.2: Price states over time for $\theta = .85$
3.2 Sequential Load Users

Current research has focused predominately on how the auction mechanism can be set up to optimize social welfare. This has led to the development and approximation of combinatorial auctions that can handle a variety of features, but often use simplistic user models. In this approach we model a user that needs to complete a certain number of jobs, \( K \), within a given deadline, \( T \), and analyze the behavior of the user as it interacts with the developing market. We assume the work a user requires must be done sequentially (not in parallel), meaning it will have at most one job served in any given hour. Additionally, we assume the user knows the statistical pricing model. In the Amazon EC2 spot instance market, the price history is given for the past 60 days and could be used to estimate the parameters of the pricing model described in the previous section. We call such a user a sequential load user and model it under two different policies. Section 3.2.1 discusses a sequential load user that follows the optimal policy for determining bid given by dynamic programming equations. Section 3.2.2 discusses a sequential load user that follows an approximate policy for determining bids given by expected rate inequalities.

3.2.1 Dynamic Programming Policy

The dynamic programming user is a sequential load user defined by a set of cost to go functions. Let \( V(i, k, t) \) be the total amount a user would expect to pay given that the current state of the price model is \( i \), the user requires \( k \) jobs, and has \( t \) time remaining. A user with no jobs remaining would stop bidding and its total expected payment would be zero, so \( V(i, 0, t) = 0 \) for all \( i \in \{0, 1, \ldots, d - 1\} \) and all \( t \geq 0 \). A user with no time remaining would expect to incur a cost dependent on how many jobs \( k \) did not get completed, so \( V(i, k, 0) = C(k) \). For simplicity we set this terminal cost as a linear function \( C(k) = \alpha k \). A user’s total expected payments for remaining time \( t \) are equal its total expected payments for remaining time \( t - 1 \) plus the amount paid for the current hour. The expected payments can therefore be determined by the dynamic programming method.
For \( i \in \{0, 1, \ldots, d-1\}, k \geq 0, t \geq 0 \) let:

\[
V(q, k, t) = \sum_{j=0}^{d-1} q_j V(j, k, t)
\]

\[
V_+(i, k, t) = \gamma_i + \theta V(i, k-1, t-1) + (1-\theta)V(q, k-1, t-1)
\]

\[
V_-(i, k, t) = \theta V(i, k, t-1) + (1-\theta)V(q, k, t-1).
\]

The quantity \( V(q, k, t) \) describes the total payment averaged over the state space. For any given \((i, k, t)\) the user can either opt to receive service at the current price or opt to not receive service at the current price and wait until the next round. The quantity \( V_+(i, k, t) \) represents the expected total payment if service was bought at price \( \gamma_i \) leading to \( k-1 \) jobs and \( t-1 \) time slots remaining one time step later. The quantity \( V_-(i, k, t) \) represents the expected total payment if service was not bought at price \( \gamma_i \) leading to \( k \) jobs and \( t-1 \) time slots remaining one time step later. We can describe the expected total payment then by:

\[
V(i, k, t) = \min\{V_+(i, k, t), V_-(i, k, t)\}.
\]

We assume all users are rational and would want service only if the expected total payment of purchasing service is less than the expected total payment of waiting until the next round. We can create a control decision \( u^*(i, k, t) \) based on this philosophy where \( u^*(i, k, t) = 1 \) if \( V_+(i, k, t) \leq V_-(i, k, t) \) or equivalently:

\[
V_+(i, k, t) - V_-(i, k, t) \leq 0,
\]

(3.1)

and \( u^*(i, k, t) = 0 \) otherwise. A control value of one implies that, given the current state \( i \), the jobs remaining \( k \), and time remaining \( t \), the user would want to purchase service at the given price \( \gamma_i \).

Based on these dynamics, it can be shown that the controls \( u^*(i, k, t) \) are non-increasing in state. Appendix A offers a proof of this and was derived in consultation with Hajek [11]. It follows from the fact that the left-hand side of (3.1) is non-decreasing in state, and therefore \( u^*(i, k, t) \) will be non-increasing in state. The following figures show the solution to the dy-
namic program for $(K,T) = (50,50)$, $\alpha = 1$, and the pricing parameters given by Table 3.2. For four of the $(k,t)$ pairs the differences in the cost to go functions, $V_+ (i,k,t) - V_- (i,k,t)$, are shown in Figure 3.3.

Table 3.2: User Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>7</td>
<td>number of price states</td>
</tr>
<tr>
<td>$q$</td>
<td>[.05,.3,.2,.05,.01,.2,.01]</td>
<td>price state distribution</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>[0.2,.25,.33,0.5,.66,.75,0.9]</td>
<td>prices as on-demand percentage</td>
</tr>
<tr>
<td>$\theta$</td>
<td>[0,1]</td>
<td>price stickiness</td>
</tr>
<tr>
<td>$K$</td>
<td>50</td>
<td>number of jobs</td>
</tr>
<tr>
<td>$T$</td>
<td>50</td>
<td>time to complete jobs</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>unfulfilled job cost</td>
</tr>
</tbody>
</table>

There are a few noteworthy points to make about Figure 3.3. As expected, these differences are non-decreasing in state. The optimal control decision described by (3.1) is to bid for any state to the left of where the plot intersects zero. For a $(k,t)$ pair in which $k << t$ this happens near the lowest price state. As $k$ moves closer to $t$ the intersection point gets pushed towards the higher price state. This makes intuitive sense from the user’s perspective. A user with a small number of jobs relative to time can afford to gamble by submitting lower bids. A user with a larger number of jobs relative to time has to bid more in order to have a higher probability of getting service. In the extreme case $(k,t) = (1,50)$ there is still a point at which the cost to go difference is negative, implying that the bidding strategy would be to bid the lowest state. As long as a bidder in this scenario has jobs remaining, it will at least bid for the lowest pricing state. In the other extreme case $(k,t) = (45,50)$ the cost to go difference still crosses zero at some point, implying that the bidding strategy would be to bid the second highest state. As long as the user’s slack, $t - k$, is strictly positive, the user should decline to receive service at the highest price.
The fact that the optimal control is non-increasing in state has an important implication. For every \((k, t)\) pair, there exists an integer threshold \(\tau^*(k, t)\) such that \(0 \leq \tau^*(k, t) \leq d - 1\) and \(u^*(i, k, t) = 1\) if and only if \(i \leq \tau^*(k, t)\). In other words, given \(k\) jobs left with \(t\) time remaining, an optimal bidding strategy of the user is to bid for the price that is given by \(\tau^*(k, t)\) and is independent of the current state. By the principle of optimality the strategy of bidders with the same \((k, t)\) values would be the same regardless of initial starting points \((K, T)\). Thus, every user of this type can be modeled with the same \(\tau^*(k, t)\) matrix.

In Figure 3.4 we show the user’s optimal strategy \(\tau^*(k, t)\) as a contour plot. The jobs and time remaining yield the optimal bidding strategy, which is represented by the degree of shading. From this graph it is easy to see that a user’s bids increase with jobs remaining \((k)\) and decrease with time remaining \((t)\). This stems from the same intuition described previously. It follows that as slack increases, the optimal bid decreases and as slack decreases, the optimal bid increases.
Figure 3.4: Bidding strategy of a dynamic programming user as a function of $k$ and $t$ for $\theta = 0$
3.2.2 Expected Rate Policy

In Section 3.2.1, the optimal strategy of sequential load users with deadlines was shown to correspond to a bidding strategy matrix $\tau^*(k,t)$ for the user independent of current price. Due to principles of optimality, in a large system of users we would only need to run the dynamic program once with the parameters of the maximum number of jobs among the users, $K_{\max}$, and the maximum time remaining among the users $T_{\max}$. From the bidding strategy matrix found with these parameters, we would have the optimal strategy for every user in the system. However as $(K_{\max}, T_{\max})$ become large, it may become unwieldy to carry around such a large matrix. The expected rate user model, introduced in this section, arises out of an attempt to simplify the strategy so that the matrix and dynamic programming equations are not needed. In this section we describe an expected rate user model where, again, we assume the user knows the pricing model.

Suppose a sequential load user wants access to a virtual machine at an expected rate, $\rho$, over a given time period $T$. The distribution of the pricing model is given by $q$. As $T \to \infty$, the user’s strategy would be to bid for the state $j$ such that $q_1 + q_2 + \ldots + q_{j-1} < \rho \leq q_1 + q_2 + \ldots + q_j$. This makes intuitive sense. The user wants to get service at a rate of $\rho$. It would bid the price such that it would expect to get service for at least that rate given by the cumulative distribution function of $q$. We can use this notion to approximate the strategy of the sequential load users given by parameters $(k,t)$. The expected rate control strategy is then based on the following: $u(i,k,t) = 1$ if $\frac{k}{t} > q_1 + q_2 + \ldots + q_{i-1}$ and $u(i,k,t) = 0$ otherwise. Since the control strategy is given by the CDF of the price distribution, which is monotonic, it is trivial to see that this control is also non-increasing in $i$. Once again, the user’s bidding strategy is independent of the current price and we can develop a strategy matrix $\tau(k,t)$. Additionally, the strategy is determined entirely by $q$ and $\gamma$. There is no dependence on the price stickiness parameter $\theta$.

Note we are still considering sequential load users. The difference is that instead of using the dynamic programming policy, users here use the expected rate policy. This is easier to implement, but slightly suboptimal. Since all the users we discuss in this thesis are sequential load users, for brevity we will distinguish between the two types of users by calling them dynamic programming (DP) users or expected rate (ER) users.
Figure 3.5 shows the expected rate bidding strategy as a contour plot under the parameters given by Table 3.2. The plot is similar to that of Figure 3.4 and with many of the same features. The user’s bid increases with the amount of jobs remaining and decreases with the amount of time remaining. By nature of the control decision, the division of contours is given by a sequence of increasing slopes corresponding to the cumulative distribution functions of $q$.

We can compare the bidding strategy of the DP user with the bidding strategy of the ER user by subtracting their respective bidding strategy matrices and observing the contour plot. Let $\delta(k, t) = \tau^*(k, t) - \tau(k, t)$. The contour plot of $\delta(k, t)$ is given by Figure 3.6 for the parameters given in Table 3.2. The figure shows strategies between the two models are extremely similar, differing at most by one state. Though there are a few bands where this does not hold, for the most part, where the two strategies differ, the DP user strategy is to bid less than the ER user. This means that there is a tendency of ER users to overbid for service.
Figure 3.6: Difference in the bidding strategy of the dynamic programming user and expected rate user
3.3 The Impact of Price Stickiness

In Section 3.1, the value of the parameter $\theta$ was shown to have a significant effect on the simulated price states. In this section we investigate the impact of $\theta$ on the dynamic programming user’s bidding strategy. The expected rate user is closely related to a value of $\theta = 0$ and we don’t consider it variable in price stickiness. Figure 3.7 shows how the expected total payments change as a result of $\theta$ under the parameters given in Table 3.2 and a $(k,t)$ pair of $(45,50)$. As $\theta$ increases, the expected total payment increases if the initial state was high and decreases if the initial state was low. Recall $\theta$ represents how likely the price model is to stay in the current state. With a high $\theta$ and a low initial state, there is a higher probability of staying in the low state and so the expected total payment should decrease. The opposite is true with a high initial state. There can also be states, however, that do not maintain monotonicity in expected total payment over $\theta$. These tend to be the states closest to the weighted average of the prices.

![Figure 3.7: Total expected payment for given initial states as a function of $\theta$ for $(k,t) = (45,50)$](image-url)
Figure 3.8 shows the bidding strategy when $\theta = .99$. The difference between this plot and Figure 3.4 is significant. As $\theta$ becomes close to one, the probability of more than one price change occurring within the time frame, $t$, becomes small. If there is no price change, then all bidding strategies would yield the same total payment. If a single price change were to occur, then the average new price would be $\bar{\gamma} = \sum_{j=0}^{d-1} q_j \gamma_j$. Recall we defined slack to be the amount of room a user has to risk not getting service, $t - k$. Under the assumption of a single price change sometime in the future, a user with positive slack would be willing to pay $\bar{\gamma}$ for service at the current time. Thus, its optimal strategy would be to bid for the state priced immediately under $\bar{\gamma}$. In Figure 3.8 the vast majority of $(k,t)$ pairs follow this strategy. As $t$ increases, the impact of $\theta$ decreases because the probability of multiple price changes occurring increases. This is why the bidding strategy under a large $t$ and small amount of slack is to bid higher than the average price $\bar{\gamma}$. Figure 3.9 shows how the optimal strategy changes over $\theta$ for various $(k,t)$ pairs. The bids of pairs with large slack tend to increase with $\theta$ and the bids of pairs with small slack tend to decrease with $\theta$. 

![Figure 3.8: Bidding strategy of a dynamic programming user as a function of $k$ and $t$ where $\theta = .99$](image)
Figure 3.9: Bidding strategy of a dynamic programming user given \((k, t)\) pairs as a function of \(\theta\)

For every \((k, t)\) pair we can also look at the average total payment previously defined as \(V(q, k, t)\). This is the weighted average of the plots given in Figure 3.7 and gives us a sense of the total amount a user would expect to pay independent of the initial price. In Figure 3.10 this plot is shown for several \((k, t)\) pairs. The figure shows that as \(\theta\) increases, the average total payment of a user increases, leading to following conjecture.

**Conjecture 3.3.1** The average total price \(V(q, k, t)\) increases in \(\theta\) for any choice of \(q, \gamma, k,\) and \(t\).

Conjecture 3.3.1 has interesting implications because it means that, no matter the model parameters, stickier prices are inherently bad for users on the average. Under higher price stickiness users would always expect to pay more.

In this section we analyzed the dependence of the optimal policy on the price stickiness parameter. It is clear that the stickiness parameter has an impact on the expected payments and bidding strategies. However, the figures in this section show this only occurs for values of \(\theta\) close to one. For most of the range \([0, 1]\) the price stickiness has little effect. Although
the price model is a simplification of Amazon’s pricing mechanism, the history of the Amazon spot prices given in [3] and [10] would yield a stickiness parameter not in the range to heavily influence a user’s behavior. Additionally, the larger the time constraints of the user, the less impact the stickiness parameter would have. Thus in the current Amazon market, it appears the user can just assume the stickiness parameter to be zero.

Figure 3.10: Average total payment for given \((k, t)\) pairs as a function of \(\theta\)
3.4 User Elasticity

In Section 3.2.1 the expected total payment when $t = 0$ is specified as a function of the number of jobs remaining, $V(i, k, 0) = C(k)$. Previous simulations have been based on a linear function $C(k) = \alpha k$ where $\alpha = 1$. However, changing the value of $\alpha$ or more generally, changing the form of $C(k)$ can alter the user’s bidding strategy. One can think of $C(k)$ as the cost to the user if it were to take its remaining jobs and buy them elsewhere. It can also be thought of as the cost to the user if the jobs were not served at all. In either case, for the linear model, it represents a price ceiling for the user. The user would never pay more than $\alpha$ in the marketplace for service.

Figures 3.11 and 3.12 show the bidding strategy of both user types under the parameters given in Table 3.2 with the exception $\alpha = 0.55$. Under this scenario, $\gamma_i < \alpha$ holds for four states $i$. In the case of the ER user shown in Figure 3.11, the plot is the same as Figure 3.5, but limited once the price state goes beyond the value of $\alpha$. Due to the dynamics of the DP user, the plot in Figure 3.12 deviates slightly from that of Figure 3.4, but the same principle behavior is observed. The user would never bid for a state beyond the value of $\alpha$. Notice again these two plots are very similar to each other. Under changing values of $\alpha$ the ER user model still closely approximates the DP user model.

As we will see in the next chapter, the values of $\alpha$ can play an important role in the elasticity of the user load in the marketplace. When the value of $\alpha$ is set to one, the load is inelastic. Valuing final costs at the on-demand price ensures that a user will complete all of its jobs within the marketplace. The closer $k$ moves to $t$, the higher the user will bid until eventually the user is bidding the on-demand price. Setting $\alpha$ less than the on-demand price introduces the possibility that the user will run out of time but still have jobs remaining. Since there is the potential for jobs to go unfulfilled, the user load can react to the price, becoming elastic.

Though the linear model of final costs is maintained throughout this thesis, we note that changing the type of function used for $C(k)$ can significantly influence bidding behavior as well. For example, a quadratic cost function would heavily punish users as the remaining number of jobs increases when time remaining hits zero. A user under this valuation would be more likely to bid higher, particularly as time remaining decreases. The cost of leaving a job unfulfilled would far outweigh the cost of receiving service in the marketplace.
Figure 3.11: Bidding strategy of a dynamic user a function of $k$ and $t$ for $\alpha = .55$

Figure 3.12: Bidding strategy of an expected rate user as a function of $k$ and $t$ for $\alpha = .55$
CHAPTER 4

SIMULATION AND RESULTS

4.1 Inelastic Load

Now that we have derived the bidding strategies of two different, but closely related, sequential load user models, we can analyze the results of a population of users following such strategies in the market. Parameters for this auction can be found in Table 4.1. We simulate over one week where each time step is considered to be one hour. At each time step new users come into the auction. The number of new arrivals each hour follows a Poisson distribution with a rate of $\lambda$. The maximum number of jobs a user requires is given by $K_{max}$ and the maximum remaining time is given by $T_{max}$. Each new user is uniformly randomly assigned a number of jobs, $k$, in the range $(0, K_{max}]$ and uniformly randomly assigned time remaining, $t$, in the range $(k, T_{max}]$. The user behavior is analyzed under both models described by Section 3.2. The stickiness parameter $\theta$ is set to zero and the terminal cost parameter $\alpha$ is set to one for all users. The price model strictly follows the given parameters and does not change as a result of demand.

Table 4.1: Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>7</td>
<td>number of price states</td>
</tr>
<tr>
<td>$q$</td>
<td>[.05, .3, .2, .05, .1, .2, .1]</td>
<td>price state distribution</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>[0.2, .25, .33, .5, .66, .75, .9]</td>
<td>prices as on-demand percentage</td>
</tr>
<tr>
<td>$\theta$</td>
<td>0</td>
<td>price stickiness</td>
</tr>
<tr>
<td>$K_{max}$</td>
<td>25</td>
<td>maximum user jobs</td>
</tr>
<tr>
<td>$T_{max}$</td>
<td>25</td>
<td>maximum user time</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>unfilled job cost</td>
</tr>
<tr>
<td>$W$</td>
<td>168</td>
<td>simulation time (hrs)</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>5</td>
<td>rate of new users</td>
</tr>
</tbody>
</table>
Figure 4.1 shows the cumulative average payment per job of the users as a function of time. It also includes the cumulative average price given by the price model. In this figure and the remaining figures in this thesis, sequential load users whose bidding strategies were determined by the dynamic programming equations given in Section 3.2.1 are labeled “DP users”. Sequential load users whose bidding strategies are determined by the policy given in Section 3.2.2 are labeled “ER users”. From Figure 4.1 we see the payments for the DP user model and ER user model are extremely close. The payments of the ER model tend to be slightly higher. This should not come as a surprise. Figure 3.6 shows there is not much difference between DP and ER bidding strategies, but that the ER user has a slight tendency to overbid. Notice that the prices paid for both of these strategies fall significantly below the average price, which would be the equivalent strategy of bidding the on-demand price under all circumstances.

![Figure 4.1: Cumulative average payment per job as a function of simulation time](image)

Figure 4.1: Cumulative average payment per job as a function of simulation time

Figure 4.2 shows the cumulative average user load over time. We observe the ER user model yields a slightly higher load than the DP user model on average. Again, this a direct result of the expected rate user having a tendency to overbid for service. With a higher bid, it is more likely for a user to get service sooner under the expected rate policy than under
the sequential load dynamic equations. Since the users are inelastic in this simulation and complete all their jobs in the marketplace, the long term average load of the two models should be the same. Figure 4.2 shows this, and the slight differences can be accounted for by the current users in the market.

Figure 4.2: Cumulative average load on the system as a function of simulation time

Figure 4.3 shows the relationship between a user’s slack and the average payment per job. The average of these is also shown and yields an inverse relationship. With a small amount of slack, there is a large variance in the prices paid and the average is fairly high. A user with small slack needs to purchase service at almost every time step. We would expect the average payment to be close to the average market price over the user’s \( t \). If the user enters the market with a large \( t \), then its payment would go towards the average price given by \( \bar{\gamma} \). However, if the user enters the market with a small \( t \), the average price over the current prices is much more variable. As slack increases, the user has time to wait at lower bidding states and becomes less dependent on the immediate market prices. As a result the variance and price decrease.
Figure 4.3: Average payment per job as a function of slack for dynamic programming users
4.2 Elastic Load

In the previous setup, by setting $\alpha = 1$ we enforce that each user completes all its jobs. The user would never encounter a situation in which $k > t$ because the optimal strategy would be to bid for the highest price state if $k = t$. This user load is elastic in the short term. From the strategy matrix $\tau^*(k, t)$ it is easy to see that as price increases, the number of users purchasing services would decrease. Due to the value of $\alpha = 1$, though, the load is inelastic in the long term. This means an increase in price would have very little impact on the load in the long term. If we were to randomly assign values of $\alpha$ less than the on-demand price, we should see the load become more elastic. Since a user no longer values unfulfilled jobs at the full on-demand price, it may be worthwhile to leave the market before completing all its jobs. The bidding strategy would be modified as described in Section 3.5.

Below are the simulation results following the same parameters given in Table 4.1 except for each user the $\alpha$ value is selected uniformly randomly on the interval of $\gamma_0$ to one. Figure 4.4 shows that the average load has decreased from Figure 4.2 and this makes sense. A bidder would never bid for a price state greater than $\alpha$, causing bidding strategies in this simulation to have generally lower bids. With lower bids, the load would decrease. Figure 4.5 shows the average pay has also decreased and for the same reasons, we would expect lower bidding strategies. This means on average a user would pay a lower amount in the market. Additionally, the cost of unfulfilled jobs, based on $\alpha$, would be less than the cost of the on-demand price used in the previous section. Finally, again, we see how close these two bidding strategies are. The cumulative average payment per job and cumulative average load of the sequential load user is almost the same as that of the expected rate user.
Figure 4.4: Cumulative average load on the system as a function of simulation time for elastic users

Figure 4.5: Cumulative average payment per job as a function of simulation time for elastic users
4.3 Load Controlled Auction

Thus far we have analyzed the users and service providers as two independent entities. The users were assumed to be price-takers and the pricing model was independent of the current demand. In reality, the interactions between the users and providers creates a feedback loop. If providers have resource constraints, then they might increase price in order to decrease the load on their systems. Eventually, a user would see these price changes as a deviation from its assumed model and reevaluate its bidding strategies under a new statistical price model. In this section we close one side of that loop. Suppose that the provider wanted to control the average load. A situation that would require this is present in the Amazon marketplace. A sudden high demand in the reserved and dedicated instances could lead to a shortage of resources available for the spot instances. In order to curb the load of the spot instances, Amazon could increase the price.

In the following simulations, the pricing mechanism changes as a result of the user load. The user, however, is ignorant of these changes (thus not completely closing the loop) and bases its strategies on the original price model. At every time step the provider observes the current load \( L(w) \) and compares it to the desired load \( \ell(w) \) where \( w \) is the simulation time. A simple proportional control mechanism is introduced in which the provider modifies the price distribution \( q \) as a result of the difference between \( L(w) \) and \( \ell(w) \). Table 4.2 shows the parameters used for this control mechanism and the simulations. The price distribution is updated every time step according to following:

\[
\begin{align*}
\beta(w) &= c(L(w) - \ell(w)) \\
q_i(w) &= q_i(w - 1)e^{i\beta(w)} \\
q &= \frac{q}{\|q\|}
\end{align*}
\]

In this simulation, the provider waits \( \delta_w \) hours to let the market initially settle before implementing the control. The parameter \( \beta(w) \) represents an update variable on the distribution \( q \) and \( \beta(w) = 0 \ \forall w < \delta_w \). If \( \beta(w) > 0 \) then the higher price states become more likely. If \( \beta(w) < 0 \) then the lower price states become more likely. In order to cut down
on some of the noise attributed to the current pricing state, the loads \( \ell(w) \) and \( L(w) \) are averaged over their previous \( v \) values.

The gain \( c \) is set such that the change in distribution \( q \) for any given time step is not too large. A large value of \( c \) would cause an over reactive response to the load. It would skew the distribution heavily towards the highest and lowest price states, such that the loads would oscillate frequently between very high and very low. A low value of \( c \) would cause a smooth but small response to the load. A value of \( \beta(w) = 0.1 \) appropriately changes distribution in a single time step. If \( q_0 = [.05, 0.3, 0.2, .05, 0.1, 0.2, 0.1] \) and \( \beta(w) = 0.1 \), then \( q_1 = [0.037, 0.245, 0.180, 0.050, 0.110, 0.243, 0.135] \), which constitutes a modest change in the price distribution. In the simulation under almost all circumstances \( |L(w) - \ell(w)| \leq 30 \), so that setting the gain \( c = \frac{1}{300} \) would yield \( |\beta(w)| \leq 0.1 \).

Table 4.2: Control Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d )</td>
<td>7</td>
<td>number of price states</td>
</tr>
<tr>
<td>( q )</td>
<td>[.05, 0.3, 0.2, .05, 0.1, 0.2, 0.1]</td>
<td>price state distribution</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>[0.2, .25, .33, 0.5, .66, .75, 0.9]</td>
<td>prices as on-demand percentage</td>
</tr>
<tr>
<td>( \theta )</td>
<td>0</td>
<td>price stickiness</td>
</tr>
<tr>
<td>( K_{max} )</td>
<td>25</td>
<td>maximum user jobs</td>
</tr>
<tr>
<td>( T_{max} )</td>
<td>25</td>
<td>maximum user time</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>variable</td>
<td>unfulfilled job cost</td>
</tr>
<tr>
<td>( W )</td>
<td>336</td>
<td>simulation time (hrs)</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>10</td>
<td>rate of new users</td>
</tr>
<tr>
<td>( \delta_w )</td>
<td>50</td>
<td>control start time</td>
</tr>
<tr>
<td>( v )</td>
<td>10</td>
<td>load averaging window</td>
</tr>
<tr>
<td>( c )</td>
<td>( \frac{1}{300} )</td>
<td>gain for load difference</td>
</tr>
<tr>
<td>( \ell(w) )</td>
<td>( 40 + 20 \sin(\frac{w - \delta_w}{80}) ) for ( w &gt; \delta_w )</td>
<td>load control function</td>
</tr>
</tbody>
</table>

In Figure 4.4 the load was shown to converge to approximately 50 jobs per hour. In this simulation the provider has soft resource constraints and wants to maintain a given load \( \ell(w) \). Figure 4.6 shows the resulting load as a function of simulation time for the users described in Section 4.2. Though noisy and delayed, the actual load follows the desired load function. Figure 4.7 shows the changes in price that have occurred in order to achieve this control. A comparison of Figures 4.6 and 4.7 shows price increases lead to a decrease in load and price decreases lead to an increase in load. This implies the users are indeed elastic.
Figure 4.6: Average load of the system as a function of simulation time for elastic users

Figure 4.7: Average payment in the system as a function of simulation time for elastic users
Figure 4.8 shows the resulting load as a function of simulation for the users described in Section 4.1. As expected, the load is initially responsive due to changes in the immediate price. However, each user has to complete all of its jobs in the market and price changes have no long-term impact on the load. The load converges to a value similar to that found in Figure 4.2. Figure 4.9 shows the average prices as a result of the price control mechanism. The provider is attempting to lower the load by increasing the price, but the load is not responding, causing the provider to further increase the price. As a result the provider skews the $q$ distribution heavily in favor of the highest price state. The user needs to complete all its jobs in the marketplace and has no choice but to pay at this inflated price.

![Figure 4.8: Average load of the system as a function of simulation time for inelastic users](image)

In this section a simple control mechanism was introduced to determine the user’s behavior under price changes. The control mechanism altered the distribution of the prices as a result of user load, closing one side of the feedback loop. The user, though, was not knowledgeable of these changes and based its bidding strategy on the original price model. As anticipated, users with a final cost parameter $\alpha$ under the on-demand price were responsive to the price changes, but users with a final cost parameter $\alpha$ equal to the on-demand were not. It should be noted that this type of control mechanism is only effective with small price stickiness. If $\theta$ were close to 1 then modifications to the $q$ distribution would not have much
effect. However, a provider that is able to control the price distribution should also be able to control the price stickiness.

Figure 4.9: Average payment in the system as a function of simulation time for inelastic users.
CHAPTER 5

CONCLUSION AND FUTURE WORK

The purpose of this thesis is to introduce a new cloud computing user model and analyze its behavior within an auction environment. The model of a user with a deadline and sequential load is developed. The optimal bidding strategy of this user is described by a set of dynamic programming equations and can be implemented without the knowledge of the current price state. An additional model of a user with an expected rate of service is developed and shown to be a close approximation to the sequential load user model. The performances of these users are shown to be similar within a second price auction closely related to the Amazon EC2 spot instance market. Finally, under variable terminal costs, the user load is shown to be elastic in that the load can be influenced by varying prices.

There is a wide variety of opportunities for future work stemming from the results found in this thesis. Often research in this area has simplified the user model, but the user’s behavior should influence the service provider’s approach to maximizing social welfare. The sequential load user model was proposed here and provides a more realistic model than a user who just bids at random. In industry, there are a myriad of different users and it would be naive to claim that this model encompasses the majority of them. Different user types should be developed and modeled to better understand behavior within the cloud computing marketplace.

An additional opportunity for future analysis is the feedback loop between the providers and the users. This thesis did not analyze a closed system, but combining optimal user models with optimal provider models could yield a better understanding of the stability of the markets. If a provider under load constraints modifies the pricing mechanism due to those constraints and the user continuously updates their optimal strategy based on the changing prices, will the system converge to some sense of equilibrium? Or does the optimality of the user conflict with optimality of the provider?
Finally, this thesis provided simulation and analysis based loosely on data from previous spot instance prices. It would be useful to analyze this user model further under a more accurate model of the Amazon EC2 spot instance market. Furthermore, verification of that analysis could be achieved by analyzing users following these theoretical strategies within the actual Amazon EC2 spot instance market.
REFERENCES


APPENDIX A

MONOTONICITY OF OPTIMAL CONTROL

Proposition .0.1 The optimal control $u^*(i, k, t)$ is nonincreasing in $i$. That is, for each pair $(k, t)$, there is an integer threshold $\tau^*(k, t)$ with $0 \leq \tau^*(k, t) \leq d$ so that $u^*(i, k, t) = 1$ if and only if $i \leq \tau^*(k, t)$.

Remark .0.1 As a result of the proposition, if the sale of service is offered as a second price auction, the user does not need to know the state of the price process, but can simply bid $\gamma_{\tau^*(k, t)}$. However, by the beginning of the next time slot, the user must learn whether a job of the user was served in the current slot, so the user knows $(k, t)$ at the beginning of each time slot.

Proof. It is required to show that if (3.1) is satisfied for some $(i, k, t)$ with $2 \leq i \leq d, k \geq 1, t \geq 0$ then it is also true with $i$ replaced by $i - 1$. A sufficient condition for that is that the left-hand side of (3.1) is nondecreasing in $i$. That is, it suffices to show that $\theta \Delta(i, k, t) \leq \gamma_i - \gamma_{i-1}$ for $2 \leq i \leq d, k \geq 1, t \geq 0$, where

$$\Delta(i, k, t) = V(i, k, t) - V(i - 1, k, t) - V(i, k - 1, t) + V(i - 1, k - 1, t).$$

The proof is completed by application of the following lemma.

Lemma .0.2 For any $t \geq 0$, $\Delta(i, k, t) \leq \gamma_i - \gamma_{i-1}$ for $2 \leq i \leq d$ and $k \geq 1$. 

36
Proof. The proof is by induction on \( t \). The base case, \( t = 0 \), is true because \( V(i, k, 0) \) does not depend on \( i \). For the induction hypothesis, suppose that \( t \geq 1 \) and that the lemma is true for \( t - 1 \). We must prove the lemma for \( t \). The proof is broken into four cases, depending on the value of \((u^{*}(i - 1, k, t), u^{*}(i, k - 1, t))\). These are the optimal decisions for the states in the two terms in \( \triangle(i, k, t) \) with minus signs. For example, case \((1, 0)\) means that it is optimal for the user to buy service if the state is \((i - 1, k, t)\) and it is optimal to not buy service if the state is \((i, k - 1, t)\). For each case, we get an upper bound on \( V(i, k, t) \) by using the control \( u^{*}(i - 1, k, t) \) for one time step, and we get an upper bound on \( V(i - 1, k - 1, t) \) by using the control \( u^{*}(i, k - 1, t) \) for one time step. We thus have

\[
V(i, k, t) - V(i - 1, k, t) \leq \begin{cases} 
\theta(V(i, k, t - 1) - V(i - 1, k, t - 1)) & \text{if } u^{*}(i - 1, k, t) = 0 \\
\gamma_i - \gamma_{i-1} + \theta(V(i, k - 1, t - 1) - V(i - 1, k - 1, t - 1)) & \text{if } u^{*}(i - 1, k, t) = 1 
\end{cases}
\]

and

\[
V(i, k - 1, t) - V(i - 1, k - 1, t) \geq \begin{cases} 
\theta(V(i, k - 1, t - 1) - V(i - 1, k - 1, t - 1)) & \text{if } u^{*}(i, k - 1, t) = 0 \\
\gamma_i - \gamma_{i-1} + \theta(V(i, k - 2, t - 1) - V(i - 1, k - 2, t - 1)) & \text{if } u^{*}(i, k - 1, t) = 1 
\end{cases}
\]

**Case (0,0)** In this case (i.e. \( u^{*}(i - 1, k, t) = 0 \) and \( u^{*}(i, k - 1, t) \)) we get an upper bound by taking the upper bound on \( V(i, k, t) - V(i - 1, k, t) \) minus the lower bound on \( V(i, k - 1, t) - V(i - 1, k - 1, t) \) to get

\[
\triangle(i, k, t) \leq \theta(V(i, k - 1, t - 1) - V(i - 1, k - 1, t - 1)) \\
- \theta(V(i, k - 1, t - 1) + V(i - 1, k - 1, t - 1)) \\
= \theta \triangle(i, k, t - 1).
\]

Using the fact \( \theta < 1 \) and the induction hypothesis, we have \( \triangle(i, k, t) \leq \triangle(t - 1, k, t - 1) \leq \gamma_i - \gamma_{i-1} \). This completes the analysis of case (0,0).
Case (0,1) Note that this case can occur only if \( k \geq 2 \). Using the same method as in Case (0,0),

\[
\begin{align*}
\triangle(i,k,t) & \leq \theta(V(i,k,t-1) - V(i-1,k,t-1)) \\
& \quad - (\gamma_i - \gamma_{i-1}) - \theta(V(i,k-2,t-1) - V(i-1,k-2,t-1)) \\
& = \theta(\triangle(i,k,t-1) + \triangle(i,k-1,t-1)) - (\gamma_i - \gamma_{i-1}) \leq \gamma_i - \gamma_{i-1}.
\end{align*}
\]

Case (1,0) Using the same method, in this case,

\[
\begin{align*}
\triangle(i,k,t) & \leq \gamma_i - \gamma_{i-1} + \theta(V(i,k-1,t-1) - V(i-1,k-1,t-1)) \\
& \quad - \theta(V(i,k-1,t-1) - V(i-1,k-1,t-1)) \\
& = \gamma_i - \gamma_{i-1}.
\end{align*}
\]

Case (1,1) Note that this case can occur only if \( k \geq 2 \). Similarly, in this case,

\[
\begin{align*}
\triangle(i,k,t) & \leq (\gamma_i - \gamma_{i-1}) + \theta(V(i,k-1,t-1) - V(i-1,k-1,t-1)) \\
& \quad - (\gamma_i - \gamma_{i-1}) - \theta(V(i,k-2,t-1) - V(i-1,k-2,t-1)) \\
& = \theta \triangle(i,k-1,t-1) \leq \gamma_i - \gamma_{i-1}.
\end{align*}
\]

Thus, the inequality \( \triangle(i,k,t) \leq \gamma_i - \gamma_{i-1} \) holds in all cases, completing the proof by induction.

\[\blacksquare\]