Dynamic Resource Allocation for Spot Markets in Cloud Computing Environments

Qi Zhang*, Quanyan Zhu‡, Raouf Boutaba*†
*David R. Cheriton School of Computer Science, University of Waterloo, Canada.
†Division of IT Convergence Engineering, POSTECH, Pohang, KB 790-784, Korea.
{q8zhang, rboutaba}@uwaterloo.ca
‡ Coordinated Science Laboratory and Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, USA. zhu31@illinois.edu

***Abstract*—The advent of cloud computing promises to provide computational resources to customers like public utilities such as water and electricity. To deal with dynamically fluctuating resource demands, market-driven resource allocation has been proposed and recently implemented by public Infrastructure-as-a-Service (IaaS) providers like Amazon EC2. In this environment, cloud resources are offered in distinct types of virtual machines (VMs) and the cloud provider runs an auction-based market for each VM type with the goal of achieving maximum revenue over time. However, as demand for each type of VMs can fluctuate over time, it is necessary to adjust the capacity allocated to each VM type to match the demand in order to maximize total revenue while minimizing the energy cost. In this paper, we consider the case of a single cloud provider and address the question how to best match customer demand in terms of both supply and price in order to maximize the providers revenue and customer satisfactions while minimizing energy cost. In particular, we model this problem as a constrained discrete-time optimal control problem and use Model Predictive Control (MPC) to find its solution. Simulation studies using real cloud workloads indicate that under dynamic workload conditions, our proposed solution achieves higher net income than static allocation strategies and minimizes the average request waiting time.**

***Keywords*-Cloud Computing; Resource Management; Model Predictive Control**

I. Introduction

As a realization of utility computing, Cloud computing aims to provide computing resources to customers like public utilities such as water and electricity. In a cloud computing environment, an Infrastructure-as-a-Service (IaaS) provider packages its physical resources (e.g. CPU, memory disk) into distinct types of virtual machines (VMs) in terms of their sizes and features, and offer them as services to the general public. For example, Amazon EC2 defines several instance types (e.g. small, large and extra large) based on their capacity in terms of CPU, memory and disk. A cloud customer, on the other hand, intends to purchase VMs to run his tasks, each of which has a specific resource requirement in terms of CPU, memory and disk. Furthermore, the utility associated with each task is captured by the price that the customer is willing to pay. Given the finite capacity for each type of resources in each data center, a fundamental problem faced by IaaS provider is how to appropriately select the price and allocate resources for each type of VM services in order to best match the interests of the customers. This problem is further complicated by the fact that demand is time varying and often has large spikes [1]. A simple and naïve solution currently adopted by most of the IaaS providers today is to specify a fixed price for each type of VM services that does not change over time. However, recent literature [2] has suggested that this flat-rate charging scheme can lead to inefficient outcomes. On one hand, when total demand is much lower than data center capacity, the data center becomes under-utilized, in which case the cloud provider can potentially lower the price to attract potential customers. On the other hand, when total demand surpasses the data center capacity due to demand spikes, it is desirable for the cloud provider to raise the price to increase revenue, while suppressing excessive demands. A common solution for this problem is to adjust the price according to supply and demand. An example of this approach has been seen in Amazon EC2 spot instance service. Specifically, Amazon EC2 creates separate resource pools and has separate capacities for each type of VMs [3]. The market price (i.e. spot price) for each VM type can fluctuate periodically to reflect the balance between demand and supply. Even though so far Amazon is the only company that offers such type of services, this issue has already received considerable attention from both industry and academia (e.g. [4], [5], [6]).

At the same time, another key issue faced by IaaS provider is energy cost of data centers. It has been reported that energy consumption constitutes more than 20% of the annual expense of a large data center [7]. A small reduction in energy consumption can save an IaaS provider millions of dollars. The most effective way of saving energy cost is to shut down unused servers [8]. This, however, requires careful capacity planning to ensure the data center does not run out of resources when demands arrive.

Combining the above observations, an IaaS provider faces the problem of dynamically adjusting the resource capacity to match resource demand from customers, in order maximize total revenue while saving energy cost. Specifically, when the total demand is low, it is desirable to reduce the data center capacity to cut down energy consumption. When the total demand exceeds data center capacity, it is desirable

to use market mechanism to ensure resources are allocated to those customers who value them the most. We call this problem the *dynamic capacity control* problem for spot markets in Cloud computing environments. We want to point out our problem is a market-based resource allocation problem, which has been studied extensively in the Grid computing literature. However, energy consumption is often neglected in the existing work. On the other hand, Our dynamic capacity control problem bears many similarities with power generator control problem in electricity spot markets [9]. The objective of the problem to control the power generation in a market setting to maximize the total revenue over time. However, there are several key differences between these two problems: Unlike electricity markets which have a single type of service (i.e. electricity), spot markets in Cloud computing typically offer multiple types of services based on multi-dimensional resource requirements. Furthermore, these sport markets must operate on the same infrastructure (i.e. the data center). These differences require an operational model different from the ones used in electricity spot markets.

In this paper, we study the dynamic capacity control problem in a single provider scenario, with the goal of dynamic adjusting the capacity of VM services to maximize the total income based on time-varying aggregate demand from customers. In our previous work [6], we have presented a solution to this problem by periodically solving a static optimization problem. However, it is known that such myopic solution (i.e. without consideration of the future) does not necessarily lead to an optimal solution over time. Furthermore, reconfiguration cost for supply adjustment have not been considered in our previous work. To address these limitations, in this paper, we present a solution using techniques from optimal control theory. Optimal control theory is a research field that specifically deals with optimization problems in dynamic settings. The standard techniques for solving this type of problems have been widely studied and used in many industries, including electricity spot markets. Specifically, we adopt the Model Predictive Control (MPC) approach to provide an online adaptive control mechanism that takes into account capacity constraints. In our approach, we first formulate the dynamic resource allocation problem as an optimal control problem. We then present an efficient solution for this problem using control theoretic techniques. Using simulations based on real cloud workloads, we show that our solution achieves high performance compared to existing solutions for this problem.

The rest of the paper is organized as follows. We first survey related research topics and results in Section II. An overview of Amazon EC2 spot instance mechanism is provided in Section III. In Section IV, we present our system model and the assumptions. After describing our model for demand response in Section V, we present our solution for the dynamic capacity control problem in Section VI.

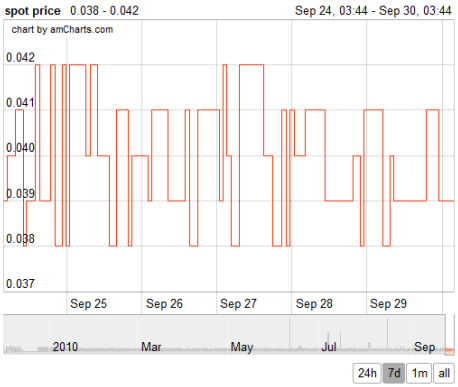


Figure 1. Price of a m1.small Linux spot instance in US-West-1 from Sept. 24-Sept. 30, 2010

Section VII is dedicated to the evaluation of the proposed solution algorithm, using realistic workload traces from Google's compute clusters. Finally, we draw our conclusions in Section VIII.

II. Related Work

There has been several recent studies on the performance of Amazon spot instance services. For instance, Andrzejak et. al. [10] studied the price and resource availability characteristics of each VM service. Yi et. al. [4] studied the problem of finding optimal checkpoint strategies to minimize the work loss due to market dynamics. These studies focus on helping end users to better use spot instance services, which is different from our objective of improving the provider's operation of spot instance services.

Using market economy to manage resource allocation has been studied extensively in the past, primarily in grid computing environments [11], [12]. The objective is to effectively provision Grid resources among a set of potentially competitive users. Various approaches, including commodity market (e.g. [13]) and auction mechanisms (e.g. [11]), have been proposed in the literature. Auction based solutions aim at achieving fair and efficient outcomes while being resilient to strategic bidding. The commodity market based approach, on the other hand, sets resource price according to supply and demand. In addition to pricing mechanisms, many market-based resource allocation systems must also address the issue of scheduling and admission control. Given limited resources, the goal of admission control is to determine what requests that should be allowed to run. The scheduling algorithms are then used to perform resource allocation to fulfill each request. Unfortunately, optimal profit-aware scheduling is generally NP-hard, and only heuristics are considered practical [14]. The authors of [14] has studied the performance of various scheduling policies for high-performance workloads. In the context of cloud computing, Stokely et. al. [15] studied market-based resource provisioning in Googles compute clusters, and presented a solution using ascending

clock auction. However, the focus of these studies is to find appropriate mechanisms to achieve desired fairness and efficiency objectives, rather than allocating resources from suppliers perspective. Furthermore, their experiments are conducted in testbeds or private environments that do not involve real currency. Furthermore, energy cost is not considered in these studies.

Another directly related research area is electricity spot markets. In these environments, the fluctuating market price is used to incentivize users to reduce their usage during peak periods in order cut-down production cost while maintaining efficient utilization of the existing infrastructure. Gallestey et. al. [16] studied the problem of determining optimal production rate of electricity generators in order to maximize immediate profits while minimize lifetime consumption of equipment. Li et. al. [17] recently proposed a solution to determine optimal market prices and demand schedules using a game-theoretic approach. Although similar in spirit, these solutions are not readily applicable to spot market in cloud computing environments as cloud resources usually have multiple types and dimensions. The service quality model is also different in the cloud computing context.

Finally, there is a large body of literature on using control theory to manage resource allocation in data center environments. For example, Kusic et. al. [18] presented a control framework for reducing energy consumption while satisfying SLA constraints. Diao et. al. [19] studied the problem of dynamically adjusting memory pool sizes for multiple agents in a database server with the goal of minimizing worst-case response time. The problem is formulated as a linear quadratic regulation problem that can be solved using standard control techniques. Our solution approach is similar to the ones described in these studies, but addresses a different problem. To the best of our knowledge, our work is the first one that studies resource allocation in market-oriented computing environments.

III. Overview of Amazon Spot Instance Mechanism

In response to the low resource utilization of computing infrastructure, Amazon EC2 has introduced the spot instance mechanism to allow customers to bid for unused Amazon EC2 capacity [20]. Currently, Amazon EC2 spot services are available for 8 types of VMs, each of which has different resource capacities for CPU, memory and disk. Amazon EC2 runs one spot market for each VM type in each availability zone. All spot markets share the free data center capacity, which is the remaining capacity after serving all the guaranteed instances[1].

To use the spot instance service, a customer submits a request that specifies the type, the number of instances, the region desired and the biding price per instance-hour. If the bidding price exceeds the current spot price, the request is fulfilled and each spot instance will run until it finishes the request or spot price exceeds the current bid. In the former case, the customer is charged for the partial-hour usage before it finishes. In the latter case, the VM is terminated without notice, and the customer is not charged for his usage. A common strategy for handling spot instance termination is to periodically save the work using progress checkpoints [4]. Notice that if a user submits a request that asks for a multitude of instances of the same type, it is possible that only a fraction of them are serviced. Hence, it is helpful to think of a multi-instance request as a set of independent single-instance requests. In addition, Amazon provides the price history to help customers decide their bids. Figure 1 shows an example of historical prices obtained from [21].

[1]Amazon currently provides 3 instance types: reserved, on-demand and spot. In this paper, the term *guaranteed instances* refer to both reserved and on-demand instances, which have guaranteed resource availability.

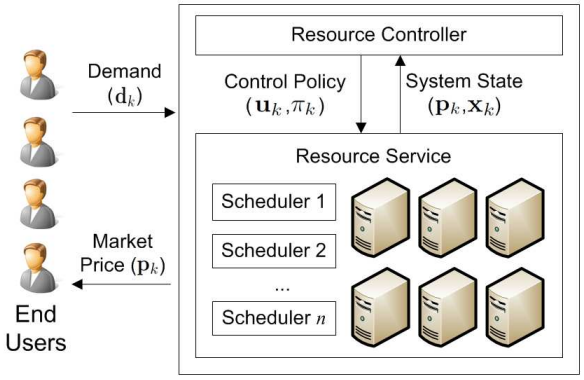


Figure 2. System Model

Generally speaking, spot instances are ideal for batch jobs that have a flexible completion time and tolerance for failures (e.g. MapReduce jobs [5]). For this type of jobs, the quality of service is determined by task wait time, which can be estimated using queueing analysis by modeling the computing system as a multi-server queue with preempt-resume priority, assuming progress checkpoints are used.

As for implementation, the spot instance pools are created from the resources which are not currently used by dedicated instances. There is a separate spot instance pool for each VM type [3]. As claimed by Amazon EC2, the spot prices fluctuate independently based their respective supply and demand. This raises the question of how to dynamically adjust resource allocaton for each VM type in order to best match the supply and the demand, given a shared resource capacity. This is the problem we address in this paper.

IV. System Model and Assumptions

The system model for dynamic capacity control problem is depicted in Figure 2. We assume there is a *Resource Controller* which is responsible for controlling both the price and and capacity allocated to each VM type. Hence the dynamic capacity control problem can be modeled as a discrete-time optimal control problem. Note that even

though both price and capacity can change over time, it is desirable that minimum number of changes made in the system. The reason is that changing price can potentially cause scheduled tasks to be de-scheduled when the market price rises over customer's bidding price. It will also reduce customer dissatisfaction due to the uncertainty and frequent oscillations in the price. On the other hand, changing capacity can potentially require migrating VMs from one to another, which will incur penalty cost. In Section VI, we will describe our cost model in details.

For the purpose of analysis, we make the following assumptions and simplifications in this paper: (1) All the machines are dedicated to spot markets, (12 the machines in the data center are identical, and (3) each machine is dedicated to a single VM type. The first assumption is made to simplify the model while capturing the essence of the problem. The second assumption is reasonable as cloud providers typically purchase large quantities of identical machines when they upgrade their data center capacities. As a result, real data centers usually consist of limited types of identical machines in terms of their hardware configurations [22]. Therefore, even if machines are not all identical in the data center, we can each type of machines separately, and have different variables for each type of machines in our formulation. The third assumption is a reasonable simplification of the real world scenario. Let $\mathcal{N} = \{1, 2, \cdots, N\}$ denote the types of VMs offered by the cloud provider. Even if there are machines that host multiple type of VMs, given a limited number of VM types $\mathcal{N}$, the total number of possible VM hosting configurations $\mathcal{C}$ (i.e. ways to allocate VMs on a physical machine) is limited. More precisely, this number is upper bounded by by $|\mathcal{C}| = n_1 \times n_2 \times ... \times n_k$, where $n_i, i = 1, \cdots, N$, denote the number of VMs of type i that can be hosted on a single machine. Therefore, the capacity allocated to each type of VMs in this case can be controlled by specifying the number of machines with each hosting configuration $c \in \mathcal{C}$. Our solution can be easily extended to handle this case.

Finally, in our formulation, we assume that the number of dedicated machines can take continuous values rather than discrete values. This assumption is reasonable as modern data centers typically contain between thousands and tens of thousands of machines [23]. Hence the weight of each individual machine in the overall solution is small. This means that given a solution of the dynamic capacity control problem consisting of continuous values, we can always round the continuous values to their nearest integer values without significantly affecting the quality of the solution.

V. Demand Modeling

An effective solution to the dynamic capacity control problem requires an accurate model of customer demand. In our case, as the resource controller adjusts price and capacity for future use, the demand model must have the capability of forecasting the behavior of future demand. Since we study the cloud provider side of the market, in this section, we will develop a model for aggregate demand rather than individual demand from each customer.

In micro-economics, demand is generally described by a demand curve that decreases monotonically with respect to the market price. In our case, for each VM type $i \in \mathcal{N}$ at time $k \in \mathbb{Z}$, we define a general function $l^i(k, p_k^i) : \mathbb{Z}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ to capture this demand curve at time k. More precisely, $l^i(k, p_k^i)$ is the number of requests of type i arrived at time k:

$$d_k^i = l^i(k, p_k^i) + v_k^i \tag{1}$$

where v_k^i is a noise function that represents the uncertainty in the demand. We assume v_k^i is zero-mean, Gaussian and mutually independent for all $k \in \mathbb{Z}$. We assume d_k^i is modeled by the provider and can be forecasted, for example, using auto-regressive (AR) functions [6].

The key challenge of using (1) to control the resource is that $l^i(\cdot)$ can be non-linear. To address this issue, we use the fact that our controller minimizes the change in price $p_k^i - p_{k-1}^i$, as mentioned in Section IV. Given a small change in price, we can approximate function locally d_k^i as a linear function of p_k^i as follows.

$$d_k^i = \bar{d}_k^i - \alpha^i(p_k^i - \bar{p}_k^i) + v_k^i \tag{2}$$

where α_k is the negative slope of the demand curve at price p_{k-1}^i at time $k-1$, $\bar{d}_k^i$ is the total demand outside the linear region, and $\bar{p}_k^i$ is price at which the linear region of the demand curve starts. In our simulation, we find this simple prediction model works effectively for estimating the demand for the purpose of dynamic resource allocation.

Define $\mathbf{d}_k = [d_k^1, ..., d_k^N]^\top$, $\bar{\mathbf{d}}_k = [\bar{d}_k^1 + \alpha^1 \bar{p}_k^1, ..., \bar{d}_k^N + \alpha^N \bar{p}_k^N]^\top$ and $\mathbf{\Psi}_k = \text{diag}\{\alpha_k^1, ..., \alpha_k^N\}$, $\mathbf{p}_k = [p_k^1, ..., p_k^N]^\top$ and $\mathbf{v}_k = [v_k^1, ..., v_k^N]$ we can write the linear demand function in the following form

$$\mathbf{d}_k = \bar{\mathbf{d}}_k - \mathbf{\Psi}_k \mathbf{p}_k + \mathbf{v}_k \tag{3}$$

Equation 3 will be used for forecasting the future demand in the design of the resource controller in the following Section.

VI. Designing a Resource Controller for the Dynamic Capacity Control Problem

At the supplier side, the cloud provider configures its resources to maximize its profits. We define a capacity constraint $C \in \mathbb{Z}$ that represents the total number of machines owned by the provider. Let $x_k^i \in \mathbb{Q} \cup [0, 1]$ be the fraction of total resources dedicated to VM type i at time k by the provider with the constraint $\sum_{i=1}^N x_k^i = 1$, for every $k \in \mathbb{K}$. As mentioned previously, we assume that the number of machines available is relatively large and x_k^i can be viewed as taking real numbers. Hence, the total number of machines dedicated to VM type i at time k can be approximately by the real value $a_k^i = Cx_k^i$. Furthermore,

we assume that each dedicated machine for type i can host b^i VMs of type i. In other words, the total capacity for type i given x_k^i dedicated machines is $c_k^i = Cb^i x_k^i$. The provider can configure his resources by transferring them among different VM types. Let $u_k^i \in [0,1]$ be the fraction of total resources added to or removed from VM type i at time k. Hence, it results in the dynamics

$$x_{k+1}^i = x_k^i + u_k^i, \quad i \in \mathcal{N}, k \in \mathcal{K} \tag{4}$$

which can be written into a compact form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{u}_k \tag{5}$$

where $\mathbf{x}_k = [x_k^1, x_k^2, \cdots, x_k^N]^\top$ and $\mathbf{u}_k = [u_k^1, u_k^2, \cdots, u_k^N]^\top$ with the capacity constraint $\mathbf{1}^\top \cdot \mathbf{x}_k \leq 1$ and $0 \leq \mathbf{u} \leq \mathbf{1}$.

Similarly, define $\pi_k^i \in \mathbb{R}_+$ as change in price at time k for type i, and denote by $\pi_k = [\pi_k^1, \pi_k^2, \cdots, \pi_k^N]'$. Hence, we also have

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \pi_k \tag{6}$$

Given a provisioned capacity c_k^i for VM type i and the length of control period T the spot instance service can be modeled as a $M/G/c$ queue with mean arrival rate $\mathbf{E}(\lambda_k^i) = \frac{1}{T}\mathbf{E}(d_k^i)$ and mean service time $S^i \in \mathbb{R}_+$. In other words, the mean service rate is $\mu^i = \frac{1}{S^i}$. Since customers are charged base resource usage, the total revenue can be expressed as the product of system utilization, price and time. On the other hand, it is also important to capture the cost of allocating capacities in our model. we assume that there is a fixed cost $e^i \in \mathbb{R}_+$ for each machine allocated to type i. In our case, e^i is the energy cost for running a machine. Therefore, the total revenue can be computed as follows:

$$\begin{aligned}
\mathbb{E}(R_k^i) &= \min\left(1, \frac{\mathbb{E}(\lambda_t^i)}{\mu^i c_k^i}\right) p_k^i T - e^i a_k^i \\
&= \min\left(1, \frac{\mathbb{E}(\lambda_k^i)}{c_k^i / S^i}\right) p_k^i T - e^i a_k^i \\
&= \min\left(Tc_k^i, \mathbb{E}(d_k^i) S^i\right) p_k^i - e^i a_k^i
\end{aligned}$$

where $\min\left(1, \frac{\mathbb{E}(\lambda_t^i)}{\mu^i c_k^i}\right)$ is the utilization of the dedicated machines for VM type i. It is easy to see that the revenue R_k^i is limited by the minimum of supply over time (i.e. Tc_k^i) and demand over time (i.e.$\mathbb{E}(d_k^i)S^i$). The optimal revenue is achieved when supply matches demand (i.e. $Tc_k^i = \mathbb{E}(d_k^i)S^i$).

However, even though it is desirable for the provider to achieve maximum utilization by exactly matching supply and demand, high utilization is generally bad for customers as it can cause significant wait time (i.e. queueing delay) for using the service To address this issue, we allow the cloud provider to maintain a desirable average request wait time $\bar{\phi}_k^i \in \mathbb{R}_+$. We assume that this wait time translates into a desired utilization level $\bar{\rho}^i \in [0,1]$. Specifically, $\bar{\rho}^i$ can be computed by solving queueing delay equation for $M/G/c$ queue [24]. Given an arrival rate λ^i and μ^i, we can compute the capacity for achieving the desired utilization as $\bar{c}_k^i = d_k^i/(\mu^i \bar{\rho}^i T)$. Hence, the provider has a second objective that is to track the desirable capacity $\bar{c}_k^i$ for reducing request waiting time.

Finally, there is a cost for adjusting the price and the capacities of all VM types. As mentioned before, price adjustment can reduce revenue and hurt customer satisfaction, while capacity adjustment incurs the cost of turning on and off machines and migrating running VMs to other machines. In our system, we define a fixed penalty cost r_1^i and $r_2^i \in \mathbb{R}_+$ associated with changes in capacity and price, respectively. Define $\sigma^i = 1/(\mu^i \bar{\rho}^i T)$ as a known constant, the objective of the capacity control problem (CC) is to minimize the cost up to a horizon K.

$$\mathbb{E}\left[\sum_{i=1}^{N}\sum_{k=1}^{\mathbf{K}} -R_k^i + q^i(b^i C x_k^i - \sigma^i d_k^i)^2 + r_1^i (u_k^i)^2 + r_2^i (p_k^i)^2\right] \tag{7}$$

Formally, define $\mathbf{Q} = \text{diag}\{q^1, ..., q^N\}, \mathbf{R}_1 = \text{diag}\{r_1^1, ..., r_1^N\}, \mathbf{R}_2 = \text{diag}\{r_2^1, ..., r_2^N\}, \mathbf{B} = \text{diag}\{\frac{1}{b^1 C}, ..., \frac{1}{b^N C}\}$, $\mathbf{E} = [Ce^1, ..., Ce^N], \mathbf{U}_k = [R_k(c_k^1) + Ce^1 x_k^1, ..., R_k(c_k^N) + Ce^N x_k^N], \mathbf{T} = \text{diag}\{TCb^1, ..., TCb^N\}$ and $\mathbf{S} = \text{diag}\{CS^1 \cdot b^1, ..., CS^N b^N\}$, we can rewrite CC in the following matrix form:

$$\begin{aligned}
\underset{\substack{\mathbf{u}_1,\ldots,\mathbf{u}_k \\ \mathbf{p}_1,\ldots,\mathbf{p}_k}}{\text{minimize}} \quad & \mathbb{E}\left[\sum_{k=1}^{K} -\mathbf{U}_k \mathbf{p}_k + (\mathbf{x}_k - \mathbf{B}\mathbf{d}_k)^\top \mathbf{Q}(\mathbf{x}_k - \mathbf{B}\mathbf{d}_k)\right. \\
& \left. + \mathbf{u}_k^\top \mathbf{R}_1 \mathbf{u}_k + \pi_k^\top \mathbf{R}_2 \pi_k - \mathbf{E}\mathbf{x}_k\right] \\
\text{subject to} \quad & \mathbf{d}_k = \bar{\mathbf{d}}_k - \mathbf{\Psi}_k \mathbf{p}_k + \mathbf{v}_k, \\
& \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{u}_k, \mathbf{p}_{k+1} = \mathbf{p}_k + \pi_k, \\
& \mathbf{U}_k \leq \mathbf{x}_k^\top \mathbf{T}, \mathbf{U}_k \leq \mathbf{d}_k^\top \mathbf{S}, \\
& \mathbf{1}^\top \cdot \mathbf{x}_k \leq 1, \\
& 0 \leq \mathbf{u}_k \leq 1, k \in \mathcal{K}
\end{aligned}$$

The above optimal control problem is a constrained linear quadratic problem (LQP) which can be solved analytically backward using dynamic programming [25]. To make the system operate in a dynamic, online setting, the resource controller can implement a MPC algorithm described by Algorithm 1. Specifically, let $\bar{\mathbf{d}}(k+i|k)$, $\mathbf{u}(k+i|k)$ and $\pi(k+i|k)$ denote the values of $\bar{\mathbf{d}}_{k+i}$, $\mathbf{u}_{k+i}$ and π_{k+i} predicted at time k respectively, based on the system model and information at time k. When the control period k starts, the controller first predicts the values of $\bar{\mathbf{d}}(k+i|k)$ for the next $i = 1, \cdots, K$ horizons using equation (3), and then solves CC to obtain the sequence of control actions $\mathbf{u}(k+i|k)$ and $\pi(k+i|k)$ for $i = 0, \cdots, K-1$. According to the standard MPC procedure, the controller will only apply the first step $\mathbf{u}(k) = \mathbf{u}(k|k)$ and $\pi(k) = \pi(k|k)$ in the sequence of control actions. This process will repeat when the next control period $k+1$ starts.

Algorithm 1 MPC Algorithm for Dynamic Capacity Control Problem

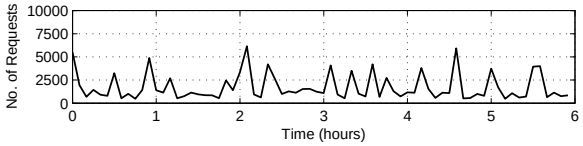
1: Provide initial state $\mathbf{x}_0$ and $\mathbf{p}_0$, $k \leftarrow 0$
2: **loop**
3: At beginning of control period k:
4: Predict $\bar{\mathbf{d}}(k+i|k)$ for horizons $i = 1, \cdots, K$ using (3)
5: Solve CC to obtain $\pi(k+i|k)$ and $\mathbf{u}(k+i|k)$ for horizons $i = 0, \cdots, K-1$
6: Change the market prices according to $\pi(k) = \pi(k|k)$
7: Change resource allocation according to $\mathbf{u}(k) = \mathbf{u}(k|k)$
8: $k \leftarrow k+1$
9: **end loop**

VII. Experiments

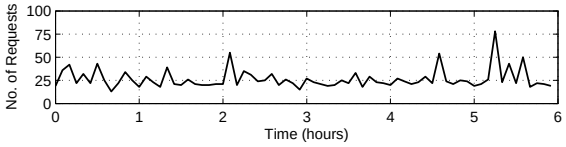
To evaluate the quality of our proposed solutions, we have implemented a prototype of spot instance system in MATLAB. Specifically, we have developed a discrete event-based VM scheduler that is controlled by our proposed Resource controller in Figure 2. To generate realistic resource requests from customers, we use the publicly available workload traces from Google's Compute Clusters [26], which describe the resource consumptions for CPU and memory of 176580 tasks [2] for a duration of over 6 hours. However, as Google's cloud is still largely private, the computer cluster traces do not contain the details of resource requests in terms of VM types and price. We artificially construct the VM types by examining the maximum resource usage of CPU and memory and match them with the available VM types offered by SpotCloud [27], a cloud computing company that provides a commodity-based market for trading computing resources. The price information is also obtained based on the existing market price in SpotCloud for VM types in the European region. Specifically, we generate bidding prices according to normal distributions with the mean values equal to the ones given by SpotCloud. By doing so, we have classified the workload into 3 types of VMs. The average bidding price, resource capacity and average running time for each VM type are summarized in Table 1. We also observed there are a few long-running tasks that persist through out the 6 hours period. For the purpose of demonstration, we focus on the tasks that arrive during the 6 hours period. The arrival rate of the tasks are illustrated in Figure 3, which is rather spiky. Furthermore, a majority of the resource requests are for small VMs. Finally, most of VMs have short running time. This observation is consistent with numerous reports on workload characterization in Cloud computing environments [23], [1].

In our simulation, we construct a medium size cluster with 7000 machines. All the machines are identical with 4 CPUs and 4 GB of memory. The control period used by the resource controller is originally set to once per hour. However, since we only have 6 hours of workload, to observe more transient behaviors of our solution algorithm, we change control period to 30 minutes. We also assume the provider would like to keep the cluster utilization around 70% in order to achieve a good balance between utilization and task wait time.

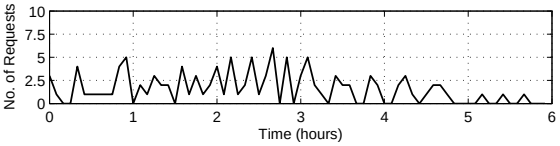
[2]A task in Google compute clusters is equivalent to a standard virtual machine.



(a) Small VMs



(b) Medium VMs



(c) Large VMs

Figure 3. Task Arrival Rate in Google's Workload Traces

Table I
Types of VMs used in the experiments

VM Type	CPU Capacity (Cores)	Memory Size (MB)	average duration (seconds)	Avg. bidding price ($)
small	1	64	1694	0.038
medium	1	128	4862	0.039
large	1	256	14049	0.041

Even though all 3 VM types share the same data center capacity, we decide to show the result for each VM type separately due to the difference in numerical scales. The capacity provisioned by the controller as well as the number of VMs running for each VM type are shown in Figures 4, 5 and 6 respectively. It can be seen that our proposed control policy gradually adjust the capacity for each VM type to best match the desired utilization level without causing severe penalties. Despite the high variability in arrival rate, the change in capacity over time seem modest, except in the beginning where there is large discrepancy between capacities and resource demands. Figure 7 shows the change in price in response to the dynamics of the arrival process. It can be seen that the price changes for all 3 type VMs are generally small. Furthermore, all three price curves show decreasing trends over time, which matches the increase in resource utilizations shown in Figures 4, 5, 6 and 8. This is because controller tries to lower the price to accommodate more demands over time. It can be also observed that the price for large VMs shows the largest fluctuation. One explanation for this observation is that the

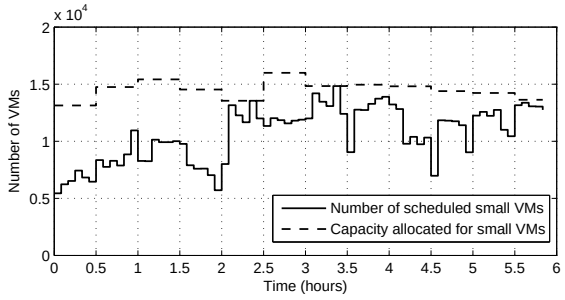


Figure 4. Number of small VMs running in the cluster

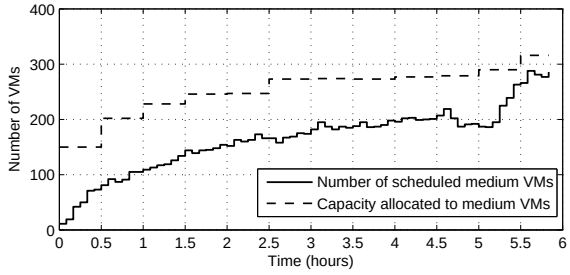


Figure 5. Number of medium VMs running in the cluster

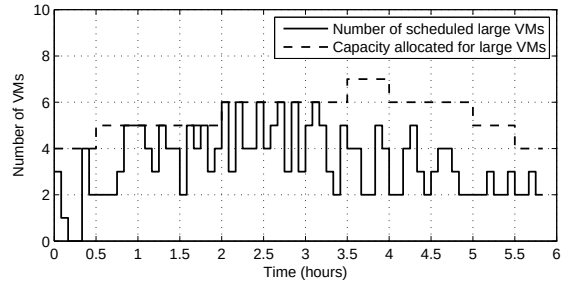


Figure 6. Number of large VMs running in the cluster

number of large VMs in the workload is rather small, hence the controller can perform fine-grained control on the prices in order to match their bidding prices.

Figure 8 shows the utilizations of allocated capacities for each VM type in each of the hours we simulated. The average utilization over 6 hours for small, medium and large VM types are 74.9%, 66.7% and 69.8% respectively. These numbers match our objective of keeping the cluster utilization around 70% for all 3 VM types. Lastly, we omit the diagram showing task wait time because over 98% of the tasks are scheduled immediately upon arrival. The remaining 2% of tasks are scheduled within the next 10 minutes.

Finally, we compare our approach with an simple strategy where there is a fixed number of dedicated machines for each VM type. The performance metrics we wish to compare are the total revenue and the average task waiting time. To make the simple strategy competitive, we computed the optimal allocation strategy in terms of number of dedicated machines for each VM type, and compare the simulation

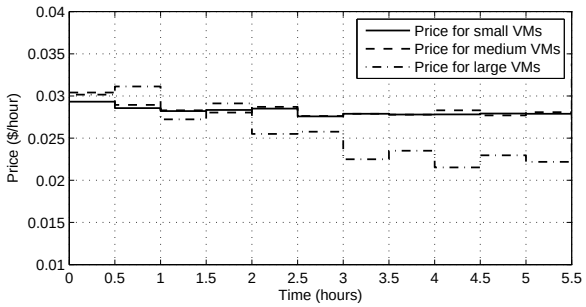


Figure 7. Prices of each type of VMs

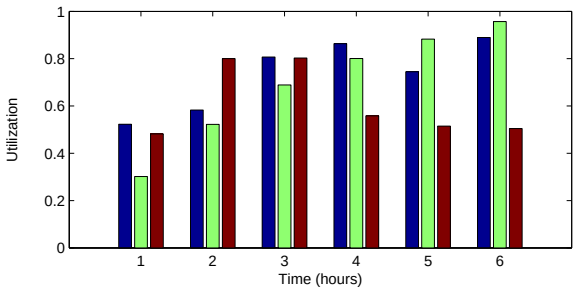


Figure 8. Utilization of allocated machines

result with our proposed solution. Note that in practice, this optimal allocation strategy is not achievable due to the requirement for future knowledge. The results using unmodified workload traces are shown in Table 2. The revenue achieved by our dynamic solution is comparable to the one obtained by the optimal allocation strategy. To make the comparison more thorough, we also modified the original workload traces and increased the arrival rate of small VMs by a factor of 3 for the period between 2 hours and 4 hours since the start time. Due to space constraint, we only summarize the revenue and average task waiting time for the modified workload traces in Table 2. It can be seen that our approach out performs the static strategy in terms of revenue and task wait time by at least 20%. This shows that our solution is most effective under highly dynamic conditions (such as flash-crowd effects) where demand may change significantly over time.

VIII. Conclusion

Using market economy to allocate resources in IaaS environments has recently received much attention, mainly because of its ability to match supply and demand and incentivize desired customer behavior. However, since cloud computing resources are offered in distinct types of virtual machines (VMs) that share the same resource capacity, it becomes a challenging problem to determine the optimal way to allocate resources to optimize total revenue while minimizing energy cost. In this paper, we study this problem in a single provider scenario motivated by the spot instance service offered by Amazon EC2. We present the

Table II
REVENUE AND TASK WAITING TIME ACHIEVED USING STATIC AND DYNAMIC ALLOCATION STRATEGIES

Workload	Strategy	Revenue ($)	Avg. task wait time (min)
Original	Static	1819.2	0.1407
	Dynamic	1808.5	0.0980
Modified	Static	2129.3	50.6450
	Dynamic	2522.8	9.5021

design of a resource management mechanism in which we can dynamically adjust both supply and price to meet customers demands, while optimizing the revenue, energy cost and request wait times. The mechanism is based on a constrained discrete-time finite-horizon optimal control formulation and we adopt MPC techniques for designing our dynamic algorithm. Our proposed solution is evaluated using realistic workload traces obtained from production clusters at Google.

There are several directions of interest to pursue in the future. First, in this paper we have exclusively studied a monopoly market scenario from a IaaS provider's perspective. It is also important to analyze the system from customer's perspective in terms of bidding behavior and service quality. Second, we would like to derive a more accurate model for demand response in public cloud environment. We are also interested in conducting more extensive experiments using workload data sets that contain price information.

ACKNOWLEDGMENT

This research was partially supported by was partially supported by the Natural Science and Engineering Council of Canada (NSERC) under its discovery program and partially by WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0).

REFERENCES

[1] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Analysis and lessons from a publicly available google cluster trace," *University of California, Berkeley, CA, Tech. Rep*, 2010.

[2] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[3] "Aws developer forums: Amazon elastic compute cloud," https://forums.aws.amazon.com/forum.jspa?forumID=30.

[4] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud," in *IEEE Int. Conf. on Cloud Comp. (CLOUD)*, 2010.

[5] Navraj Chohan et. al., "See spot run: Using spot instances for mapreduce workflows," in *USENIX HotCloud*, 2010.

[6] Q. Zhang, E. Gurses, R. Boutaba, and J. Xiao, "Dynamic resource allocation for spot markets in clouds," in *Proceedings of the USENIX HotICE workshop*, 2011.

[7] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, 2010.

[8] L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen, "Greencloud: a new architecture for green data center," in *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*. ACM, 2009, pp. 29–38.

[9] E. Gallestey, A. Stothert, M. Antoine, and S. Morton, "Model predictive control and the optimization of power plant load while considering lifetime consumption," *IEEE Transactions on, Power Systems*, vol. 17, no. 1, pp. 186–191, 2002.

[10] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecomm. Sys.*, 2010.

[11] Brent Chun, et. al., "Mirage: a microeconomic resource allocation system for sensornet testbeds," in *Proc. of the 2nd IEEE workshop on Embedded Networked Sensors*, 2005.

[12] C. Weng et. al., "An economic-based resource management framework in the grid context," in *ACM/IEEE CCGrid*, 2008.

[13] R. Wolski, J. Plank, T. Bryan, and J. Brevik, "G-commerce: Market formulations controlling resource allocation on the computational grid," *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2001.

[14] F. Popovici and J. Wilkes, "Profitable services in an uncertain world," *ACM International Conference on High Performance Computing, Networking and Storage (SC'05)*, 2005.

[15] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a market economy to provision compute resources across planet-wide clusters," in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2009.

[16] E. Gallestey, A. Stothert, M. Antoine, and S. Morton, "Model predictive control and the optimization of power plant load while considering lifetime consumption," *IEEE Transactions on Power Systems*, vol. 17, no. 1, pp. 186–191, 2002.

[17] N. Li, L. Chen, and S. H. Low, "Optimal demand response based on utility maximization in power networks," in *IEEE Power and Energy Society General Meeting*, July 2011.

[18] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[19] Y. Diao, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano, "Using mimo linear control for load balancing in computing systems," in *IEEE American Control Conference (ACC)*, 2004.

[20] "Amazon ec2 spot instances," http://aws.amazon.com/ec2/spot-instances/.

[21] "Cloud exchange." http://www.cloudexchange.org/.

[22] B. Sharma, V. Chudnovsky, J. Hellerstein, R. Rifaat, and C. Das, "Modeling and synthesizing task placement constraints in google compute clusters," *2nd ACM Symposium on Cloud Computing (SOCC'11)*, 2011.

[23] A. Mishra, J. Hellerstein, W. Cirne, and C. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[24] "Multi-machine systems (lecture notes on queueing theory)," http://www.win.tue.nl/ iadan/sdp/h11.pdf.

[25] T. Başar and G. Olsder, *Dynamic noncooperative game theory*. Society for Industrial Mathematics, 1999, vol. 23.

[26] "Google cluster data," http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[27] "Spotcloud - cloud capacity clearing house / spot market: Home," http://www.spotcloud.com.