Dynamic Resource Allocation for Spot Markets in Clouds

Qi Zhang, Eren Gürses, Raouf Boutaba
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1
{*q8zhang, egurses, rboutaba*}*@uwaterloo.ca*

Jin Xiao
IT Convergence Engineering
POSTECH
Pohang, South Korea
jinxiao@postech.ac.kr

Abstract

Cloud computing promises on-demand provisioning of resource to applications and services. To deal with dynamically fluctuating resource demands, market-driven resource allocation has been proposed and recently implemented by commercial cloud providers like Amazon EC2. In this environment, cloud resources are offered in distinct types of virtual machines (VMs) and the cloud provider runs a continuous market-driven mechanism for each VM type with the goal of achieving maximum revenue over time. However, as demand of each VM type can fluctuate independently at run time, it becomes a challenging problem to dynamically allocate data center resources to each spot market to maximize cloud provider's total revenue. In this paper, we present a solution to this problem that consists of 2 parts: (1) market analysis for forecasting the demand for each spot market, and (2) a dynamic scheduling and consolidation mechanism that allocate resource to each spot market to maximize total revenue. As optimally allocating resources for revenue maximization is a NP-hard problem, we show our algorithms can approximate the optimal solutions to this problem under both fixed and variable pricing schemes. Simulation studies confirm the effectiveness of our approach.

1 Introduction

Cloud computing aims at providing computing resources as public utilities like water and electricity. In a cloud computing environment, resources are typically offered in distinct types of VMs that a customer can purchase on-demand. Traditionally, cloud providers specify a fixed price for each type of VM offerings. However, it has been shown that this pricing scheme is often inefficient due to lack of incentives to rationalize demand. 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 wishes to encourage customers to submit more requests. On the other hand, when total demand rises over the data center capacity, it is desirable for the cloud provider to incentivize the customers to reduce their demand. A promising solution to this problem is to use market economy to reshape the demand by dynamically adjusting the price of each VM type. Specifically, when total demand is high, the mechanism raises the price to ensure resources are allocated to users who value them the most. When total demand is low, the mechanism lowers the prices and provides incentive for customers to increase their demand. Market-driven resource allocation has been applied to Grid computing environments in the past [5, 6]. Recently, it has also been adopted to cloud computing. In December 2009, Amazon EC2 launched its spot instance service to sell its unused data center capacity using a market-driven resource allocation mechanism. Even though this is only the first step towards a full-fledged market-driven cloud service, it has already received considerable attention from both industry and academia (e.g. [12, 9]).

Amazon EC2's spot instance mechanism shares many similarities with the standard uniform price auction mechanism: The provider assigns resources to bidders in decreasing order of their bids until all available resources have been allocated or all resource requests have been satisfied. The selling price (i.e. the spot price) is equal to the lowest winning bid. It is known that a sealbid uniform price auction is a truthful auction, providing the supply level is adjustable *ex post* (i.e., after the bids have been decided) [4]. Dynamic supply adjustment also prevents small size collusions that been observed for Internet auctions such as eBay auction. Recently, the optimal supply adjustment problem for a single uniform price auction has been studied [4].

However, as multiple spot markets operate on a shared resource pool in each data center, a critical question arises regarding how to best distribute data center capacities to each individual spot market. A naive solution is to

employ a static allocation strategy that pre-computes the resource allocation to each spot market. There are several drawbacks to this approach. First, the free capacity of a data center can change due to dynamic conditions such as machine failure. Second, as spot markets are designed to handle fluctuating demands, a static allocation strategy can lead to situations where certain spot markets are over-supplied while some others are under-supplied. While over-supplying resources to a market may lead to wasted resources, under-supplying resources can lead to revenue loss. In both cases, a static allocation strategy may lead to sub-optimal outcomes. Therefore, it is important to dynamically adjust supply for each spot market based on current market situation, so that resources are allocated to users who value them the most.

In this paper, we study the problem of dynamic resource allocation for simultaneous spot markets. Our goal is to allocate free data center capacity to each spot market in a timely manner that maximizes the total revenue. Our solution approach consists of 2 parts: (1) Modeling market demand using time-series analysis and forecast the future demand, and (2) Dynamic pricing and scheduling spot requests to maximize the expected revenue over time. In particular, we consider both fixed and variable pricing schemes. As the optimal resource allocation problem is a variant of the NP-hard *multiple knapsack problem (MKP)*, we show our algorithms can approximate the optimal solutions to this problem under both schemes. The actual performance of our algorithms are demonstrated using simulations.

This paper is organized as follows: We provide an overview of Amazon's spot instance mechanism in Section 2, Section 3 describes our proposed dynamic resource allocation framework. We describe the 2 steps of our solution, namely demand prediction and dynamic resource allocation in Section 4 and 5 respectively. Section 6 presents our simulation results, Section 7 discusses related work and Section 8 concludes the paper.

2 Overview of Amazon Spot Instance Mechanism

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

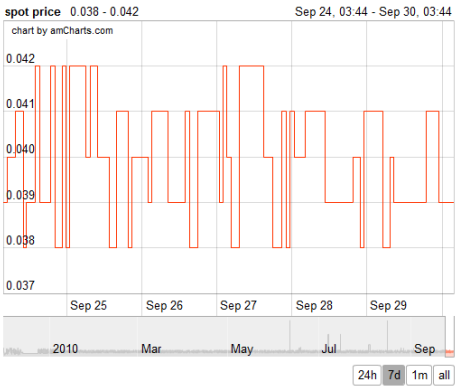


Figure 1: Price of a m1.small linux spot instance in US-West-1 from Sept. 24-Sept. 30, 2010

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 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 during the partial hour. A common strategy for handling spot instance termination is to periodically save the work using progress checkpoints [12]. Notice that if a user submits a request that asks for many instances of the same type, it is possible that only a fraction of them is satisfied. 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 price history graph obtained from [2].

Amazon's spot instance mechanism can be described as a continuous seal-bid uniform price auction, where identical goods are sold at identical price. It is known that a single round seal-bid uniform price auction is a truthful mechanism if the supply level is adjustable [4]. Therefore, each player's optimal strategy is to report its true valuation in its request. Identical price ensures fairness of the auction outcome, and truthfulness ensures that Amazon can adjust the supply to maximize revenue.

On the implementation side, Amazon's spot instance mechanism operates in a continuous fashion. A spot instance can start running as soon as the request is submitted and bidding price is higher than the current spot price. This can be implemented by having the instance with high bidding price to preempt the instance with a low bidding price, when there is no capacity for schedul-

[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.

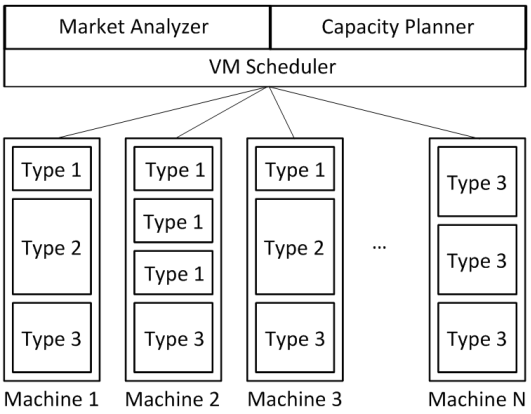


Figure 2: System Model

ing both instances. The spot price may be adjusted when there are no instances running at the current price.

However, as multiple spot markets share the same resource pool, it is unclear how to best differentiate different instance types to maximize the total revenue. This is the problem we try to address in this paper.

3 A Dynamic Resource Allocation Framework for Simultaneous Cloud Markets

Our proposed dynamic resource allocation framework for multiple simultaneous spot markets is shown in Figure 2. Each physical machine run multiple types of VM instances, some are guaranteed instances while others are spot instances. At run-time, the *Market Analyzer* periodically analyzes the market situation and forecast the future demand and supply level. Specifically it predicts the future demand curve as well as supply level (i.e. free capacity) over time period $[t_0, t_0 + T]$ at sampling interval Δt. Based on the prediction, the *Capacity Planner* decides the expected price of each type of VM in each market. This allows the *VM Scheduler* to make online scheduling decisions for revenue maximization. The detailed implementation of each component is described in the sections below.

4 Demand Modeling and Prediction

The Market Analyzer is required in our framework as we try to adjust price of each spot market for future demands. More specifically, we want to predict the demand curves in time period $[t_0, t_0 + T]$ for each spot market, where t_0 denotes the current time and T denotes the prediction period. At any time t between t_0 and $t_0 + T$, a demand curve can be constructed capturing the relationship between quantity of acceptable requests and bidding price, as shown in Figure 3. Let p_i denote ith possible

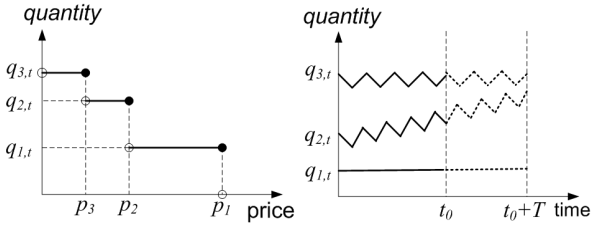


Figure 3: Example demand curve at time t and over time

bidding price in decreasing order of their values, and $d_{i,t}$ denote the corresponding requests that bid at price p_i at time t, and $q_{i,t}$ denote the demand that bids at price *at least* p_i at time t. According to the spot market mechanism, if the spot price p_s is set to p_i, then we are allowed to schedule $q_{i,t}$ VM requests. Generally speaking, $q_{i,t}$ is a non-increasing function of p_i.

Our approach for predicting the expected demand curve is as follows: Recall the bidding price of each individual VM request is truthful and independent of the current market situation, consequently we can model the demand quantity $d_{i,t}$ independently for each p_i. Hence we use a time-series method to forecast the future demand $d_{i,t}$ for each possible value of p_i for $t_0 \leq t \leq t_0 + T$ using past demand history. Forecasting future demand in general has been studied extensively in both economics and computer systems [11]. We adopt a simple autoregressive (AR) model. It estimates the value $d_{i,t}$ using the historical values $d_{i,t-1}, ..., d_{i,t-k}$ as:

$$d_{i,t} = \sum_{j=1}^{k} \phi_j d_{i,t-j} + \epsilon_t$$

where $\phi_1, \phi_2, ..., \phi_k$ constitute a set of parameters for historical values, and ϵ_t is uncorrelated white noise with mean 0 and standard deviation σ_ϵ. All of the above parameters can be computed from historical demand data. Using the AR model, we can compute an expected demand curve for the next period $[t_0, t_0 + T]$. AR model and its more general forms (i.e. ARMA and ARIMA) [11] are well established techniques in the literature.

We believe an AR model is appropriate for several reasons. First, comparing to machine-learning based approaches such as neural networks, AR is light-weight and easy to implement. Second, comparing to markovian models which only capture long-term trends, an AR model is capable of capturing short-term trends, which is important for predictive resource allocation.

5 Dynamic Revenue Maximization for Multiple Cloud Markets

The estimated supply curve allows us to design algorithms to achieve trade-offs between different spot mar-

kets. The main objective of the problem is to schedule requests of each spot market to maximize the expected revenue over the next prediction period, without exceeding capacities of individual machines.

We shall present our solution approach for two different pricing schemes: In the *fixed pricing scheme*, price of a VM type does not vary with the current supply and demand. The solution for this scheme is already applicable for scheduling guaranteed instances. On the other hand, in the *uniform pricing scheme*, the price of a VM type is adjustable at run-time. For the case, we present an algorithm that estimates the best price for maximizing the expected total revenue. Together with scheduling algorithm proposed for the fixed pricing scheme, the framework is expected to maximize total revenue over time.

5.1 Dynamic Scheduling and Server Consolidation for Fixed Pricing Scheme

In the fixed pricing scheme, each VM type has a fixed price that does not fluctuate with the current supply and demand. The VM revenue maximization problem (VRMP) in this case can be modeled as a MKP as follows: Given a set of machines M and D resource types (e.g. CPU, memory and disk), where each machine $m \in M$ has a capacity c_{dm} for each resource type $d \in D$. There is a set of VMs V to be scheduled. Each VM i has a size a_{id} for each $d \in D$ and a value v_i. The objective is to schedule a set of VMs to maximize the total value, as represented by the following Integer Program (IP):

$$\begin{aligned} \max & \sum_{i \in V} \sum_{m \in M} v_i x_{im} & \\ \text{subject to} & \sum_{i \in V} a_{id} x_{im} \leq c_{dm} & \forall m \in M, d \in D \\ & x_{im} \in \{0,1\} & \forall i \in V, m \in m \end{aligned}$$

MKP is an NP-hard combinatorial optimization problem. Our solution to this problem is based on a $\frac{1}{2} - \epsilon$ local search algorithm given in [8]. As depicted by Algorithm 1, the algorithm proceeds in rounds. In each round, if there exists a potential new configuration for a single machine m by scheduling, migrating and preempting VMs, the scheduler will try to carry out this scheduling operation. Our algorithm stops when no operation can improve the current solution. To achieve fast convergence rate, We require each local search operation to improve solution quality for each machine m by at least $(1 + \epsilon)$. This cause the algorithm to lose an approximation factor of ϵ. This algorithm can be used both as a scheduling and a server consolidation algorithm, as it simply tries to maximize the values of VMs scheduled on each machine. To minimize the disruption, the scheduler first tries to schedule all VMs using available free capacity. When the total demand is reaching data center capacity, the scheduler will start performing preemption and migration.

Algorithm 1 Local Search Approximation Algorithm $Local(P)$ for VRMP

1: **for** $\forall m \in M$ **do**
2: Find a set of VMs S' that among pending requests and the current running VMs on the machine m, maximize the total value $R(S') = \sum_{j \in S'} v_j$.
3: **end for**
4: **while** $\exists$ a machine m such that $R(S') \geq R(S)$ **do**
5: Schedule the requests in $S' \backslash S$, preempt and migrate VMs in $S \backslash S'$ if necessary.
6: **end while**

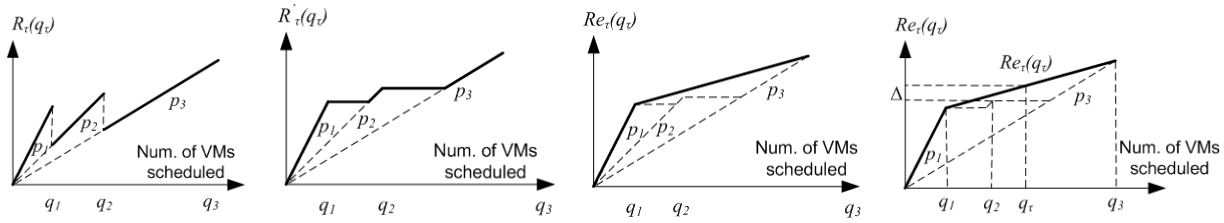
5.2 Price Estimation for the Uniform Pricing Scheme

We now consider the case where price of each instance vary with the demand curve and the resource availability. Formally, this dynamic revenue maximization with variable price (DRMVP) problem is identical to VRMP except that individual VMs no longer have a fixed price. Rather, the price is determined by the estimated demand curve $R_\tau(q_\tau)$. This can be modeled by the following IP:

$$\begin{aligned} \max & \sum_{\tau \in T} R_\tau(q_\tau) & \\ \text{subject to} & \sum_{m \in M} x_{\tau m} = q_\tau & \forall \tau \in T \\ & \sum_{\tau \in T} a_{\tau d} x_{\tau m} \leq c_{md} & \forall m \in m, d \in D \\ & x_{\tau m}, q_\tau \in \mathbb{N} \cup \{0\} & \forall m \in m, \tau \in T \end{aligned}$$

This program is more difficult to solve than the VRMP, as the objective function is non-linear. Specifically, the revenue function $R_\tau(q_\tau)$ for a single VM type is a piecewise linear function. Figure 4(a) illustrates $R_\tau(q_\tau)$ using the example given in Figure 3, where the slope of each linear segment is equal to p_1, p_2 and p_3 respectively. It can be observed that in some situations, scheduling a VM can cause the current market price to be lowered, resulting in a sharp drop in total revenue. Even though we can define $R'_\tau(q_\tau) = \max_{q \leq q_\tau} \{R_\tau(q)\}$ and use $R'_\tau(q_\tau)$ instead in DRMVP (since $R'_\tau(q_\tau)$ is an achievable revenue function by reducing number of scheduled instances), the function $R'_\tau(q_\tau)$ is still non-linear. The simplest solution to problem is to try every possible combination of prices and solve each case independently. However, this can lead to a large number of cases when the number of possible prices is large.

Motivated by similar work on market clearing algorithms for piecewise linear objective revenue functions



(a) $R_\tau(q_\tau)$ (b) $R'_\tau(q_\tau)$ (c) $Re_\tau(q_\tau)$ (d) An example illustrating Lemma 1

Figure 4: Revenue vs Supply Curve For a Single Market

[10], our approach to deal with this issue is to approximate $R'_\tau(q_\tau)$ using a *concave envelope* function $Re_\tau(q_\tau)$. Essentially, $Re_\tau(q_\tau)$ is computed by constructing a upper convex hull using the extreme points in $R'_\tau(q_\tau)$. It is possible to prove the following property for $Re_\tau(q_\tau)$:

Lemma 1. *$Re_\tau(q_\tau) \leq 2 \cdot R'_\tau(q_\tau)$ for any q_τ.*

Proof. Consider any q_τ. Assume $Re'_\tau(q_\tau)$ is on a linear segment with extreme points (q_i, Re_i) and (q_j, Re_j) where $i \leq j$. By definition of convex hull, Re_i and Re_j are also points on R'_τ. (This is illustrated in Figure 4(d), where $i = 1$ and $j = 3$.) Let their corresponding slope (i.e. unit price) in R'_τ be $p_{i\tau}$ and $p_{j\tau}$ respectively. Clearly, $p_{i\tau} \geq p_{j\tau}$, $Re_\tau(q_i) = q_i \cdot p_{i\tau}$ and $Re_\tau(q_j) = q_j \cdot p_{j\tau}$. The slope p_s of the segment of $Re_\tau(q_\tau)$ can be bounded by $p_s = \frac{q_j \cdot p_{j\tau} - q_i \cdot p_{i\tau}}{q_j - q_i} \leq p_{j\tau}$. At point $(q_\tau, Re_\tau(q_\tau))$, the total overestimate of $Re_\tau(q_\tau)$ can be upper bounded by $\Delta = Re_\tau(q_\tau) - R'_\tau(q_\tau) \leq (q_\tau - q_i) \cdot p_s \leq q_\tau \cdot p_{j\tau}$. This is because $R'_\tau(q_\tau)$ is non-decreasing between q_i and q_j. Now, the actual value of $R'_\tau(q_\tau)$ can be lower bounded by $q_\tau \cdot p_{j\tau}$, as when $q_\tau \leq q_{j\tau}$ the unit price is at least $p_{j\tau}$. Combining the arguments, we get $Re_\tau(q_\tau) - R'_\tau(q_\tau) \leq R'_\tau(q_\tau)$. The lemma follows. □

Lemma 1 essentially suggests that we use the concave envelope function $Re_\tau(q_\tau)$ to approximate piecewise linear function $R'_\tau(q_\tau)$, losing an factor at most 2. The key advantage of using a concave objective function is that now we can now treat each individual VM request separaterly. Define $v_{q\tau} = Re_\tau(q_\tau) - Re_\tau(q_\tau - 1)$ as value for scheduling the qth request of type τ. Since $Re_\tau(q_\tau)$ is concave, $v_{q\tau}$ is a non-increasing function of q. This definition of $v_{q\tau}$ is similar to the concept of *marginal value* in economics. Given the value of each request, we can now construct an instance VRMP' where each VM has value according to differentials of $Re_\tau(q_\tau)$.

Theorem 1. *Running Algorithm 1 using $v_{q\tau}$ defined above is a $\frac{1}{4} - \epsilon$ approximation algorithm for DRMVP.*

Proof. First, it is evident that the optimal solution of DRMVP has a corresponding solution in VRMP' with at least the same revenue, as $Re_\tau(q_\tau) \geq R'_\tau(q_\tau)$. Now, we show that every solution of VRMP' has a corresponding solution in DRMVP with at least half the total revenue. Indeed, as values of VMs are non-increasing, any solution of VRMP' that schedules q_τ VMs of type τ has a corresponding solution in DRMVP that schedules the q_τ most valuable VMs with at least $\frac{1}{2}$ the revenue, according to Lemma 1. As Algorithm 1 is a $\frac{1}{2}$ approximation algorithm of VRMP', the theorem is proven. □

Theorem 1 provides a worst bound on the performance of the algorithm. Our simulation shows the algorithm often performs quite well, even though VRMP and DRMVP are only the abstract representation of the actual scheduling problems. As for implementation, the capacity planner is responsible for setting the minimum price for the next period using the algorithm in Theorem 1. The scheduler runs Algorithm 1 for scheduling VM instances at run-time.

6 Experiments

We have implemented a prototype of our framework using CloudSim [3], a Java based simulator for simulating Cloud computing environments. We simulated a 1000 machine data center capable of hosting 8 instance types available in Amazon EC2. The spot requests for each VM type arrive according to a non-homogenous poisson process that may have artificial high and arrival rate periods (i.e. demand spikes and valleys). The bidding prices and task durations for each VM type are generated from a normal distribution. As for scheduling policies, we implemented both static allocation policy where resource assignment on each machine is pre-computed (by solving a knapsack problem), and the dynamic allocation policy according to Algorithm 1. In our simulation, spot prices are recomputed once per hour. We define the *income rate* as the sum of prices of all the scheduled requests. Figure 5 compares the income rate of the two policies for a duration of 16 hours, when the average arrival rate is steady over time. The dynamic allocation policy (i.e. Algorithm 1) slightly outperforms static policy (about 10% gain), as it considers multiple machine configurations compared to a single configuration that the static policy uses. This performance gain is

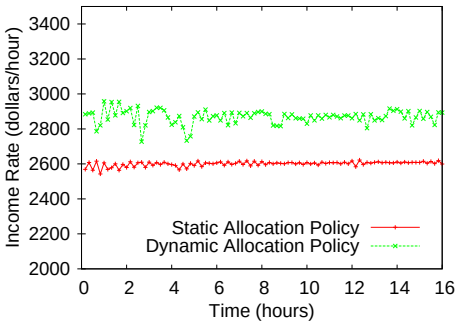


Figure 5: Income rate with static demands

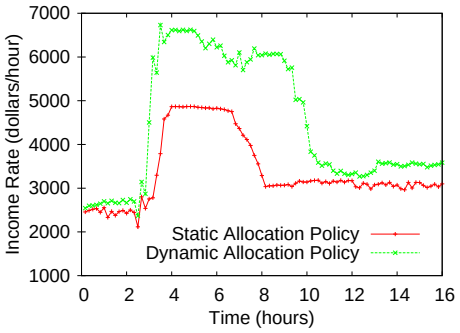


Figure 6: Income rate with fluctuating demands

amplified when the demand pattern changes over time, as shown in Figure 6. This is because the capacity planner is able to adjust the price according to the demand pattern. Finally, although our algorithm does not consider the revenue loss due to VM preemption, we have computed the total revenue loss according to Amazon's pricing policy. The average results are shown in Table 1. It can be observed that the dynamic allocation policy causes more revenue loss, but this negative impact is outweighed by the positive revenue gain. It is part of our future work to refine our algorithms to consider VM migration and preemption cost.

7 Related Work

Market-driven resource allocation has been studied in the past, particularly for Grid Computing environments [5, 6]. However, the existing work do apply to Cloud computing as Grid Computing does not use real currency. Furthermore, most of the scheduling algorithms for Grid and parallel computing focus on minimizing makespan rather than maximizing revenue. We show in this paper that maximum profit scheduling can be formulated as a dynamic MKP. When the price is adjustable, similar knapsack problems with piece-wise linear objective functions have been studied previously [10], but with different objectives. Finally, even though MKP has been studied extensively for several decades, devising efficient approximation algorithms for MKP is still a challenge.

Table 1: Average revenue achieved by different policies

Policy	Metric	Income	Revenue Loss	Net Income
Static	Mean	67030.44	399.01	66631.43
	Std.	13573.32	172.45	13400.87
Dynamic	Mean	78026.33	3398.36	74627.97
	Std.	15173.28	1083.63	14089.65

Cherkuri et. al. gave a polynomial time approximation algorithm with approximation guarantee $(1-\epsilon)$ [7], but the running time is prohibitive even for large ϵ. For efficient algorithms, linear programming based solutions can achieve an approximation algorithm of $\frac{e-1}{e}$ [8], but it is not adaptive to dynamic conditions. The local search algorithm described in [8] seems to be the most appropriate solution for dynamic resource allocation.

8 Conclusion

We have presented a cloud management framework that dynamically allocates data center resources to spot markets to maximize cloud provider's total revenue. Specifically, We designed efficient algorithms for scheduling VM requests under both fixed pricing scheme and uniform pricing scheme. Our experiments confirm the effectiveness of our approach. In the future, we plan to further improve our algorithms by considering the revenue loss due to VM preemption and migration.

References

[1] Amazon ec2 spot instances. http://aws.amazon.com/ec2/spot-instances/.

[2] Cloud exchange. http://www.cloudexchange.org/.

[3] Cloudsim. http://www.cloudbus.org/cloudsim/.

[4] AMIR DANAK, S. M. Resource allocation with supply adjustment in distributed computing systems. In *International Conference on Distributed Computing Systems (ICDCS)* (2010).

[5] BRENT CHUN, ET. AL. Mirage: a microeconomic resource allocation system for sensornet testbeds. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors* (2005).

[6] C. WENG ET. AL. An economic-based resource management framework in the grid context. In *ACM/IEEE CCGrid* (2008).

[7] CHEKURI, C., AND KHANNA, S. A ptas for the multiple knapsack problem. In *ACM Symp. on Discrete Algorithms* (2000).

[8] LISA FLEISCHER ET. AL. Tight approximation algorithms for maximum general assignment problems. In *ACM Symposium on Discrete Algorithms* (2006).

[9] NAVRAJ CHOHAN ET. AL. See spot run: Using spot instances for mapreduce workflows. In *USENIX HotCloud Workshop* (2010).

[10] S. KAMESHARWAN ET. AL. Nonconvex piecewise linear knapsack problems. In *Euro. Jnl. of Operational Research* (2009).

[11] WILLIAM W. S. WEI. Time series analysis: univariate and multivariate methods. In *Addison Wesley* (1990).

[12] YI, S., KONDO, D., AND ANDRZEJAK, A. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *IEEE Int. Conf. on Cloud Computing (CLOUD)* (2010).