2015 IEEE 8th International Conference on Cloud Computing

Cost-Minimizing Online VM Purchasing for Application Service Providers with Arbitrary Demands

Shengkai Shi*‡, Chuan Wu*, and Zongpeng Li†
*The University of Hong Kong, †University of Calgary, ‡Ecosystem and Cloud Services Business Group, Lenovo
Email: shishengkai0624@gmail.com, cwu@cs.hku.hk, zongpeng@ucalgary.ca

***Abstract*—Recent years witness the proliferation of Infrastructure-as-a-Service (IaaS) cloud services, which provide on-demand resources (CPU, RAM, disk) in the form of virtual machines (VMs) for hosting applications/services of third parties. Given the state-of-the-art IaaS offerings, it is still a problem of fundamental importance how the Application Service Providers (ASPs) should rent VMs from the clouds to serve their application needs, in order to minimize the cost while meeting their job demands over a long run. Cloud providers offer different pricing options to meet computing requirements of a variety of applications. However, the challenge facing an ASP is how these pricing options can be dynamically combined to serve arbitrary demands at the optimal cost. In this paper, we propose an online VM purchasing algorithm based on the Lyapunov optimization technique, for minimizing the long-term-averaged VM rental cost of an ASP with time-varying and delay-tolerant workloads, while bounding the maximum response delay of its jobs. In stark contrast with the existing studies, the proposed algorithm enables an ASP to optimally decide the amount of reserved, on-demand and spot instances to purchase simultaneously. Rigorous analysis shows that our algorithm can achieve a time-averaged resource cost close to the offline optimum. Trace-driven simulations further verify the efficacy of our algorithm.**

I. Introduction

As a major type of cloud services, Infrastructure-as-a-Service (IaaS) cloud offerings provide abundant and elastic computing resources for third party usage, which has remarkably revolutionized the way of enabling scalable and dynamic Internet applications. More and more Application Service Providers (ASPs) are launching their applications in clouds, without the need for building and maintaining their owned IT infrastructures. The leading online content provider Netflix [1] offers on-demand Internet video service and receives enormous streaming requests from its worldwide subscribers every minute. With Amazon EC2 [2], Netflix can run critical encoding tasks, to serve its clients' video demands, with a number of VM instances configured with the selected encoding software, and shut them down when completed [3].

Cost management is still a crucial task in such a switch to cloud-based services. VM instance purchases from the clouds play a critical role for cost management of an ASP. Under the pay-as-you-go pricing model, a practical problem for ASPs is how to minimize the VM purchasing cost while guaranteeing a good service performance. Cloud vendors usually offer multiple pricing options that allow the flexibility to optimize costs [4]. The commonly adopted cloud pricing schemes are (1) reserved instance pricing, (2) on-demand instance pricing, and (3) spot instance pricing. With reserved instances, users pay an one-time upfront fee and reserve instances with a significantly lower hourly charge for a 1-year or 3-year term. On-demand instances enable users to pay a fixed hourly rate with no long-term usage commitment. Spot instances, offered by Amazon EC2 [2], allow users to bid whatever price they want for spare instances with no upfront commitment and to run them at an hourly rate substantially lower than the on-demand rate, whenever their bid price is larger than the spot market price. A pricing example of reserved instance, on-demand instance and spot instance is given in Table I.

TABLE I
PRICING OF RESERVED INSTANCE, ON-DEMAND INSTANCE AND SPOT INSTANCE (LINUX, US WEST) IN AMAZON EC2, AS OF APR 25, 2015.

Instance Type	Pricing Option	Up-front	Hourly
m3.medium	1-year reserved	$372	$0.0425
	on-demand	$0	$0.070
	spot	$0	$0.008
m3.large	1-year reserved	$751	$0.0857
	on-demand	$0	$0.140
	spot	$0	$0.0255

While reserved instances are more beneficial for applications with long-term steady or predictable workload, on-demand instances are more recommended for applications with short-term spiky or unpredictable workload. Spot instances could act as complements of reserved and on-demand instances to achieve further cost savings. However, the spot price is fluctuating all the time in tune with the demand and supply levels. Fig. 1 shows a significant variation in Amazon EC2 spot prices for Linux/UNIX instances of type r3.xlarge from Mar 27, 2015, to Apr 24, 2015. We observe that at times the spot price exceeds even the on-demand price. Thus purchasing spot instances risks frequent job interruptions during the execution, and is more suitable for time-flexible or delay-tolerant applications. Simply operating the entire workload with only one pricing option can be highly cost-ineffective. Since gaps do exist among different pricing schemes, it is quite desirable yet challenging for ASPs to intermingle different pricing options based on their own demand, in order to optimize the long-term-averaged cost. In particular, with time-varying demands, one ASP should answer two basic questions at any decision-making instant: (1) how many instances to purchase, and (2) which type to purchase (reserved, on-demand, spot or all)?

This paper was supported in part by grants from Hong Kong RGC under contracts HKU 717812 and 718513.

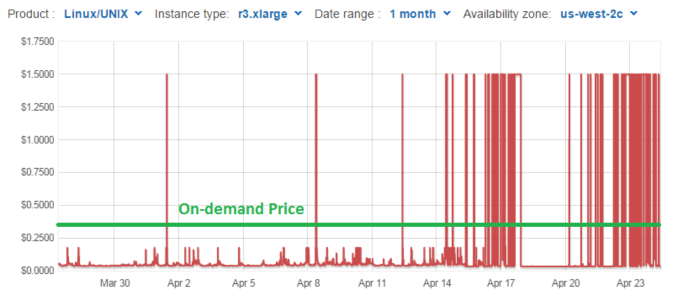

Fig. 1. The variation in Amazon EC2 spot prices for Linux/UNIX r3.xlarge instances in the US-West-2c region from Mar 27, 2015, to Apr 24, 2015.

Amazon EC2 introduced one customized service, AWS Trusted Advisor, to help users realize more cost reduction [5]. The AWS Trusted Advisor can make recommendations on whether the customer can save money with a more suitable VM instance, by drawing on the previous usage data aggregated across all consolidated billing accounts with complex data mining and machine learning techniques. There have been some efforts on IaaS cost management in terms of uncertain demand [6][7], but they require *a priori* knowledge of the workloads or accurate prediction of future information. Even though some statistics may be obtained using dynamic programming techniques, it suffers from high computation complexity, and hence not suitable for online decision making in practice [8][9]. In addition, very few studies have addressed the randomness of the spot prices, and more importantly, how to achieve cost optimization under this price uncertainty.

To the contrast, we seek to integrate all available pricing schemes and design effective online algorithms for the long-term operation of ASPs. We formulate the long-term-averaged VM cost minimization problem of an ASP with time-varying and delay-tolerant workloads in a stochastic optimization model. An efficient online VM purchasing algorithm is designed to guide the VM purchasing decisions of the ASP based on the Lyapunov optimization technique. Lyapunov optimization provides a framework for designing algorithms with performance arbitrarily close to the optimal offline performance over a long run of the system, without the need for any future information. It has been widely applied to resource allocation optimization in data centers [10][11]. Different from these existing studies, our online VM purchasing algorithm does not require any prior knowledge or assume any distribution of the workload. Moreover, it addresses the possible job interruption due to uncertain availability of spot instances. To our best knowledge, this work is the first effort on jointly leveraging all three common IaaS cloud pricing options, in order to exploit the highly-coveted cost advantages of cloud computing.

The rest of the paper is organized as follows. We review related literature in Sec. II, describe the system model in Sec. III, design the online algorithm in Sec. IV, present the simulation results in Sec. V, and conclude the paper in Sec. VI.

II. Related Work

We now provide a snapshot of the related work. There has been a growing interest in cost management of clouds. Sharma *et al.* [12] propose a cost-aware capacity provisioning mechanism for ASPs to choose server configurations and reconfigurations in order to minimize the rental cost of cloud infrastructure. Qiu *et al.* [13] propose an optimization framework for dynamic, cost-minimizing migration of content distribution services into a hybrid cloud infrastructure that spans geographically distributed data centers. Khanafer *et al.* [14] model the cost optimization problem of a cloud file system as a variant of classical Ski-Rental problem, and propose new randomized algorithms to generate significant cost savings. Setty *et al.* [15] provide a cost-effective resource provisioning scheme for deploying publish/subscribe services in the cloud, so as to minimize the total cost of VM acquisition and bandwidth consumption, while ensuring a good subscriber satisfaction. Roh *et al.* [16] formulate a concave game taking into account both the resource pricing of clouds and resource competition of ASPs, and investigate the characteristics of the equilibrium point.

There have been some works discussing the strategic combination of different cloud pricing options. Leslie *et al.* [17] and Lu *et al.* [18] exploit cost-effective hybrid resource provisioning approaches for deploying applications on on-demand and spot VMs. Menache *et al.* [19] propose an online learning algorithm for allocating on-demand and spot VM instances for batch applications. The candidate policy weights are dynamically adjusted through learning from performance on job executions, spot prices and workload characteristics, in order to reinforce best performing policies. Hong *et al.* [7] study the costs of margins, which are a pool of servers kept active for unpredictable potential workload surges, and propose a dynamic programming approach for minimizing the margin cost. Then a VM purchasing strategy combining reserved and on-demand VMs is designed to achieve optimal true cost. However, the proposed approaches are possible only when *a priori* knowledge of the workload is available. Zhao *et al.* [6] analyze the time-varying spot prices in Amazon EC2 and show that the spot price is highly unpredictable. Then a stochastic resource rental planning model is proposed to take spot price uncertainty into account, and a hybrid VM renting approach based on on-demand and spot VM instances is provided. The demand in the planning horizon is simply assumed to be known. Wang *et al.* [9] design an instance reservation strategy via a cloud brokerage mechanism to minimize the total cost of both reserved and on-demand VM instances. A cloud broker can aggregate the demands from a large number of users to smooth out individual demand bursts, and time-multiplex partial usage in the same instance-hour. The proposed method suffers from the prohibitive complexity of dynamic programming and relies on workload prediction. Based on [9], Wang *et al.* [8] further propose a deterministic algorithm and a randomized online reservation algorithm blending the two pricing options without any knowledge of future demand. However, spot VMs are not leveraged for achieving more cost reductions and more flexibility in VM acquisition. Also in general, these schemes do not provide any tradeoff guarantee between the cost and the service performance. Our design addresses these issues.

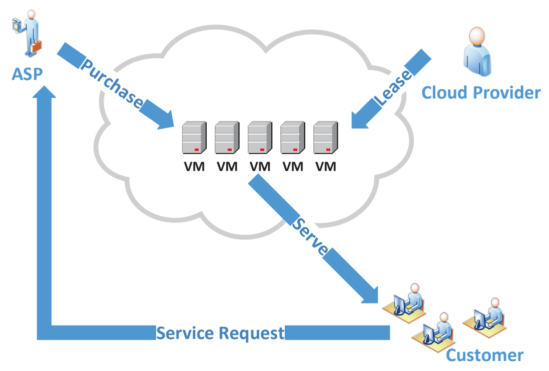

Fig. 2. System overview.

III. System Model

As illustrated in Fig. 2, we consider an ASP providing services to its customers over the Internet. Instead of using its own resources, the ASP accepts and processes job requests with VMs purchased from an IaaS cloud provider. The system runs in a time-slotted fashion with time slots of equal lengths indexed by $t = 0, 1, \ldots$, where each t is a decision-making instant of the ASP. In practice, a slot t could be one hour.

A. Job Model

There are total G types of jobs served by the ASP. Each job, or service request from a customer of the ASP, is characterized by a three-tuple (s_g, l_g, m_g). Here, $s_g \in [1, S]$ represents the type of the required VM, where S is the maximum number of VM types, and each type comprises a different configuration of CPU, memory, and storage; l_g is the Service Level Agreement (SLA) of the type-g job, specified by the maximum response delay for scheduling a job, *i.e.*, the time-span from when the job arrives to when it is dropped or starts to run on the scheduled VM; m_g is the number of time slots required and is referred to the workload of one type-g job. During its execution, a job could be suspended and resumed later.

B. Scheduling Model

A FIFO queue is maintained at the ASP to store the unscheduled workload, Q_g, for each job type $g \in [1, G]$. In every time slot, jobs arrive at the ASP. Let $r_g(t) \in [0, r_g^{max}]$ denote the number of type-g jobs that arrive at the beginning of each time slot t, where r_g^{max} is the maximum value of $r_g(t)$. We assume that $r_g(t)$ is an i.i.d stochastic process across time slots. Upon arrival of $r_g(t)$ type-g jobs, $m_g r_g(t)$ units of workload are appended to Q_g. The ASP then decides how to distribute the awaiting jobs. We use $u_g(t)$ to represent the number of type-g jobs successfully scheduled in time slot t. When a type-g job is scheduled for processing, the job departs from its queue and starts to run on a type-s_g VM. Let $u_g(t^-)$ denote the number of type-g jobs scheduled before t, which are still running at t. For each newly scheduled type-g job at t or each leftover type-g job at t, one unit workload is deducted from Q_g at the end of time slot t. When a job's maximum response time cannot be met, it is dropped. Let $D_g(t) \in [0, D_g^{max}]$ be the number of type-g jobs dropped by the ASP in time slot t, where D_g^{max} is the maximum number of type-g jobs allowed to be dropped in one time slot. The drop of $D_g(t)$ type-g jobs introduces $m_g D_g(t)$ units of workload reduced from Q_g. Denote $Q_g(t)$ as the total unscheduled workload of type-g jobs in time slot t. Thus, it evolves over time following the dynamics specified by

$$\begin{aligned} Q_g(t+1) =& max\{Q_g(t) - u_g(t) - u_g(t^-) - m_g D_g(t), 0\} \\ &+ m_g r_g(t), \end{aligned} \tag{1}$$

assuming that the initial queue is empty, *i.e.*, $Q(0) = 0$.

In order to satisfy the SLA constraint, we define the following virtual queues, each associated with a workload queue Q_g. We use the $\epsilon-$persistence queue technique [20] to create virtual queue $Z_g(t)$, which starts from $Z_g(0) = 0$ and evolves as

$$\begin{aligned} Z_g(t+1) = \max\{&Z_g(t) + \mathbf{1}_{\{Q_g(t)>0\}} \cdot [\epsilon_g - u_g(t) - u_g(t^-)] \\ &- m_g D_g(t) - \mathbf{1}_{\{Q_g(t)=0\}} u_g^{max}, 0\}, \forall g \in [1, G]. \end{aligned} \tag{2}$$

Here, ϵ_g is a constant that ensures $Z_g(t)$ grows whenever there is unscheduled workload in Q_g, and u_g^{max} is the maximum number of type-g jobs that could be processed simultaneously by the ASP, with $0 \leq u_g(t) + u_g(t^-) \leq u_g^{max}$. Indicator function $\mathbf{1}_{\{Q_g(t)>0\}}$ is 1 when $Q_g(t) > 0$ and 0 otherwise. Similarly, $\mathbf{1}_{\{Q_g(t)=0\}}$ is 1 when $Q_g(t) = 0$ and 0 otherwise. Length of a virtual queue reflects the cumulated response delay of workloads from the respective workload queue. If the system can bound the maximum lengths of the workload queues and virtual queues with properly set ϵ_g, then the maximum response delay of jobs can be bounded.

C. VM Provisioning Model

The system deploys a hybrid VM provisioning mechanism through an integration of three different pricing options: (1) reserved instances, (2) on-demand instances, and (3) spot instances. After observing the VM demands of all jobs being processed, including newly scheduled jobs and leftover jobs, the ASP makes a decision to reserve $a_s(t) \in [0, a_s^{max}]$ type-s VMs in time slot t (*i.e.*, reserved instances), $\forall s \in [1, S]$, where a_s^{max} is the reservation limit of type-s instances per time slot. Let N denote the fixed number of time slots any reserved VM is reserved for (*i.e.*, the reservation period). Each reserved type-s instance will stay effective in the whole reservation period $[t, t+N-1]$. So the total number of type-s reserved VMs remaining effective at t is $\sum_{\tau=t-N+1}^{t} a_s(\tau)$. The aggregated reserved resource may be insufficient to accommodate all job demands. The ASP may also launch additional $b_s(t) \in [0, b_s^{max}]$ on-demand type-s VMs, $\forall s \in [1, S]$, to serve the residual demand, where b_s^{max} is the on-demand limit of type-s instances per time slot.

Spot VMs can also be launched as an alternative to reserved or on-demand VMs. After observing the current spot price, the ASP bids for spot instances and use them whenever its willingness-to-pay is larger than the spot price. How to set the bid prices depends on the ASP's evaluation on a number of operational factors. In this paper, we set the bid price for type-s spot VMs as the price of type-s on-demand VMs. The rationale is that it is more cost-efficient to launch spot VMs only when the current spot price is smaller than the current on-demand price. Let $f_s(t) \in [0, f_s^{max}]$ denote the number of type-s spot VMs the ASP obtains in time slot t, where f_s^{max} is the upper limit. Different from on-demand and reserved instances, spot instances are priced in real-time as illustrated in Fig. 1. So the ASP should always prepare for the possibility that its spot instances are terminated when the spot prices exceed the bid prices. Since the bid price is the same for each type-s spot VM started at the same time slot, all type-s spot VMs acquired at t will be terminated once the spot price exceeds the bid price. We consider that the interrupted jobs due to terminated spot VMs will be served immediately by newly launched on-demand VMs. In case that an out-of-bid event occurs for type-s spot VMs during time slot t, $f_s(t)$ on-demand VMs will be instantly purchased to carry on the leftover workloads of the interrupted jobs. We assume that the switch time from terminated spot VMs to newly launched on-demand VMs is negligible.

At each time slot, the total supply of VMs purchased should always accommodate the total demand of job scheduling:

$$\begin{aligned}&\sum_{\tau=t-N+1}^{t} a_s(\tau) + b_s(t) + \mathbf{1}_{\{\beta_s > \gamma_s(t)\}} f_s(t) \\ &\geq \sum_{g:s_g=s} [u_g(t) + u_g(t^-)], \forall t, \forall s \in [1, S]. \qquad (3)\end{aligned}$$

IV. Dynamic Cost Minimizing Algorithm

A. Problem Formulation

A number of cost parameters are associated with the VM cost optimization. When dropping one job, a penalty is enforced to compensate for the customer's loss. Let σ_g denote the penalty to drop one type-g job. For reserved instances, a user needs to make a one-time payment for the reservation, and the charging policies slightly differ across different types of reserved instances [21]. We limit our discussions to reserved instances with fixed costs, which represent a common case in IaaS clouds. The total cost of a reserved instance is abstracted as a one-time upfront reservation fee, which is denoted as α_s for a type-s reserved VM.

We assume that the length of each time slot t matches the duration of one billing cycle for on-demand and spot instances, *e.g.*, one hour. Let β_s denote the price of running one type-s on-demand VM per billing cycle. To model the availability of spot VMs during one time slot, we use $P_s(t) \in [0, 1]$ to denote the probability that a termination event may happen within $[t, t+1]$ for the type-s spot VMs acquired at t, which could be dynamically estimated based on historical price series, within a window of past time slots of a certain size. Let $\gamma_s(t)$ denote the spot price at t of running one type-s spot VM for one billing cycle. We consider two cases: (1) Case 1: a type-s spot VM successfully runs for one entire time slot and is charged at $\gamma_s(t)$ for the billing cycle $[t, t+1]$, which happens with probability $1 - P_s(t)$. (2) Case 2: a type-s spot VM is terminated during $[t, t+1]$ with probability $P_s(t)$, and is replaced immediately by a newly launched type-s on-demand VM to process the leftover workloads. Spot instances will not be charged for any partial billing cycle of usage, while partial billing cycle consumed by on-demand instances is charged as a full billing cycle. Thus the expected VM cost in Case 2 should be $[1 - P_s(t)]\gamma_s(t) + P_s(t)\beta_s$.

Given the system model and cost model, the total VM cost to accommodate all demand in time slot t is given by

$$\begin{aligned}Cost(t) = &\sum_{s\in[1,S]} \{\alpha_s a_s(t) + \beta_s b_s(t) \\ &+ \mathbf{1}_{\{\beta_s > \gamma_s(t)\}}[P_s(t)\beta_s f_s(t) \\ &+ (1 - P_s(t))\gamma_s(t) f_s(t)]\} + \sum_{g\in[1,G]} D_g(t)\sigma_g. \qquad (4)\end{aligned}$$

The time-averaged expected VM purchasing cost is

$$Cost_{av} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} E[Cost(t)]. \qquad (5)$$

Therefore, the VM cost minimization pursued by the ASP can be formulated as follows

$$\begin{aligned}\min \quad & Cost_{av} & (6)\\ s.t. \quad & 0 \le r_g(t) \le r_g^{max}, \forall g \in [1, G], \forall t; & (7)\\ & 0 \le u_g(t) + u_g(t^-) \le u_g^{max}, \forall g \in [1, G], \forall t; & (8)\\ & 0 \le D_g(t) \le D_g^{max}, \forall g \in [1, G], \forall t; & (9)\\ & 0 \le a_s(t) \le a_s^{max}, \forall s \in [1, S], \forall t; & (10)\\ & 0 \le b_s(t) \le b_s^{max}, \forall s \in [1, S], \forall t; & (11)\\ & 0 \le f_s(t) \le f_s^{max}, \forall s \in [1, S], \forall t; & (12)\\ & \text{Constraints (1)-(3),} &\end{aligned}$$

where $u_g^{max} = N * a_{s_g}^{max} + b_{s_g}^{max} + f_{s_g}^{max}$. The objective of our optimization problem is to make dynamic job scheduling and VM acquiring decisions, so as to minimize long-term time-averaged VM cost. Important notations are summarized in Table II, for the ease of reference.

B. Online Algorithm Design

We now design an online algorithm to solve the cost minimization problem in (6). To minimize the time-averaged objective function in (6) based on decisions at each time slot, we resort to the drift-plus-penalty framework in Lyapunov optimization [20], a classical technique for transforming a long-term time-average optimization problem into a series of similar one-shot optimization problems. In each time slot t, we have a set of queues $\Theta(t)$ as

$$\Theta(t) = \{Q_g(t), Z_g(t) | g \in [1, G]\}. \qquad (13)$$

Define Lyapunov function as follows

$$L(\Theta(t)) = \frac{1}{2} \sum_{g\in[1,G]} [(Q_g(t))^2 + (Z_g(t))^2], \qquad (14)$$

TABLE II
IMPORTANT NOTATIONS

s_g	type of the VM required by one type-g job
m_g	number of time slots required by one type-g job
l_g	maximum response delay for scheduling one type-g job
$r_g(t)$	number of type-g jobs arriving at the beginning of t
$D_g(t)$	number of type-g jobs dropped by the ASP at t
$u_g(t)$	number of type-g jobs scheduled for processing at t
$u_g(t^-)$	number of running type-g jobs left over before t
$Q_g(t)$	length of workload queue for type-g jobs at t
$a_s(t)$	number of new type-s reserved VMs at t
N	reservation period for reserved VMs / length of a super time frame
$b_s(t)$	number of launched type-s on-demand VMs at t
$f_s(t)$	number of obtained type-s spot VMs at t
α_s	upfront reservation fee for one type-s reserved VM
β_s	price for running one type-s on-demand VM per billing cycle
$\gamma_s(t)$	spot price at t of running one type-s spot VM for one billing cycle
σ_g	penalty for dropping one type-g job
$P_s(t)$	estimated probability of a termination event in time span [t, t+1] for type-s spot VMs obtained at t

where the constant $\frac{1}{2}$ is added for the convenience of mathematical derivations. Next, we define the one-slot conditional Lyapunov drift as follows

$$\Delta(\Theta(t)) = E[L(\Theta(t+1)) - L(\Theta(t))|\Theta(t)]. \tag{15}$$

Following the framework of Lyapunov drift-plus-penalty algorithm, we add the VM purchasing cost as a penalty function to obtain the drift-plus-penalty term

$$\Delta(\Theta(t)) + VCost(t), \tag{16}$$

where $V > 0$ is a user-defined positive constant that can be understood as the weight of the VM purchasing cost in this expression, and can be tuned to different values to indicate the tradeoff between the cost and the SLA guarantee. The following lemma defines such an upper bound on this drift-plus-penalty term.

Lemma 1: Let $V > 0$, and let $\epsilon_g > 0$. Then the drift-plus-penalty term satisfies the following inequality:

$$\begin{aligned}
&\Delta(\Theta(t)) + VCost(t) \leq \\
&B + \sum_{g\in[1,G]} Q_g(t)[m_g r_g(t) - u_g(t) - u_g(t^-) - m_g D_g(t)] \\
&+ \sum_{g\in[1,G]} Z_g(t)[\epsilon_g - u_g(t) - u_g(t^-) - m_g D_g(t)] \\
&+ V \sum_{s\in[1,S]} \{\alpha_s a_s(t) + \beta_s b_s(t) \\
&+ \mathbf{1}_{\{\beta_s > \gamma_s(t)\}}[P_s(t)\beta_s f_s(t) + (1 - P_s(t))\gamma_s(t) f_s(t)]\} \\
&+ V \sum_{g\in[1,G]} D_g(t)\sigma_g,
\end{aligned} \tag{17}$$

where $B = \frac{1}{2}\sum_{g\in[1,G]}\{[\epsilon_g]^2 + (m_g r_g^{max})^2 + 2[u_g^{max} + m_g D_g^{max}]^2\}$.

Proof: Detailed proof is provided in [22].

Different from previous work using Lyapunov optimization [10][11], we seek to model the more general scenario in which a job may take more than one time slot to finish and the ASP can choose any pricing option at every time slot. In each time slot t, the ASP observes the queues $Q_g(t)$ and $Z_g(t)$, and the number of left-over jobs $u_g(t^-)$, and then decides the optimal values of $D_g(t)$, $u_g(t)$, $a_s(t)$, $b_s(t)$ and $f_s(t)$ to minimize the the upper bound shown on the right-hand side of inequality (17). We can get the following one-shot optimization problem to be solved by the ASP in each time slot t:

$$\begin{aligned}
\min \quad & \varphi_1(t) + \varphi_2(t) \\
s.t. \quad & \text{Constraints (1)-(3), (8)-(12),}
\end{aligned} \tag{18}$$

where

$$\varphi_1(t) = \sum_{g\in[1,G]} D_g(t)[V\sigma_g - m_g Q_g(t) - m_g Z_g(t)]$$

$$\begin{aligned}
\varphi_2(t) = V &\sum_{s\in[1,S]} \{\alpha_s a_s(t) + \beta_s b_s(t) \\
&+ \mathbf{1}_{\{\beta_s > \gamma_s(t)\}}[(1 - P_s(t))\gamma_s(t) f_s(t) + P_s(t)\beta_s f_s(t)]\} \\
&- \sum_{g\in[1,G]} u_g(t)[Q_g(t) + Z_g(t)].
\end{aligned}$$

After careful derivation, minimization problem (18) can be decoupled to two independent optimization problems dealing with (a) job dropping, and (b) job scheduling and VM purchasing, respectively.

(a) Job Dropping. The number of dropped jobs $D_g(t)$, $\forall g \in [1, G]$, is obtained by solving the following optimization problem:

$$\begin{aligned}
\min \quad & D_g(t)[V\sigma_g - m_g Q_g(t) - m_g Z_g(t)] \\
s.t. \quad & \text{Constraint (9).}
\end{aligned} \tag{19}$$

The optimal solution of problem (19) is:

$$D_g(t) = \begin{cases} D_g^{max} & \text{if } Q_g(t) + Z_g(t) > \frac{V\sigma_g}{m_g} \\ 0 & \text{if } Q_g(t) + Z_g(t) \leq \frac{V\sigma_g}{m_g} \end{cases}.$$

It indicates that one type-g job with more dropping penalty and less workload is less likely to be dropped.

(b) Job scheduling and VM purchasing. The decisions on the number of jobs to schedule, the number of reserved VMs to purchase, the number of on-demand VMs to launch, and the number of spot VMs to acquire, all at t, can be obtained by solving the following optimization problem:

$$\begin{aligned}
\min \quad & V \sum_{s\in[1,S]} \{\alpha_s a_s(t) + \beta_s b_s(t) \\
& + \mathbf{1}_{\{\beta_s > \gamma_s(t)\}}[(1 - P_s(t))\gamma_s(t) f_s(t) \\
& + P_s(t)\beta_s f_s(t)]\} - \sum_{g\in[1,G]} u_g(t)[Q_g(t) + Z_g(t)]
\end{aligned} \tag{20}$$

$$s.t. \quad \text{Constraints (3)(8)(10)(11)(12).}$$

Problem (20) is a joint job scheduling and VM purchasing problem. We can start with solving $u_g(t)$ by assuming already known feasible assignments to $a_s(t)$, $b_s(t)$, and $f_s(t)$. To minimize (20), we should maximally schedule jobs of type-g_s^*, whose observed value of $Q_g(t) + Z_g(t)$ is the largest among all types of jobs requiring type-s VMs. We have

$$g_s^* = argmax_{g:g_s=s}[Q_g(t) + Z_g(t)], \forall s \in [1, S]. \quad (21)$$

The number of type-g_s^* jobs we can schedule at t is decided by constraint (3), at

$$\begin{aligned} u_{g_s^*}(t) =& a_s(t) + b_s(t) + \mathbf{1}_{\{\beta_s > \gamma_s(t)\}} f_s(t) \\ &- \sum_{g:s_g=s} u_g(t^-) + \phi_s(t). \end{aligned} \quad (22)$$

Here, $\phi_s(t) = \sum_{\tau=t-N+1}^{t-1} a_s(\tau)$. Except type-$g_s^*$ jobs, no other types of jobs are scheduled, *i.e.*,

$$u_g(t) = 0, \forall g \neq g_s^*, \forall s \in [1, S]. \quad (23)$$

Hence the second part of (20) can be expressed using variables $a_s(t)$, $b_s(t)$, and $f_s(t)$. Removing the constants, (20) can be converted to the following equivalent VM purchasing problem

$$\begin{aligned} \min \quad & \sum_{s\in[1,S]} \{a_s(t)[V\alpha_s - Q_{g_s^*}(t) - Z_{g_s^*}(t)] \\ & + b_s(t)[V\beta_s - Q_{g_s^*}(t) - Z_{g_s^*}(t)] \\ & + \mathbf{1}_{\{\beta_s > \gamma_s(t)\}} f_s(t)[V(1 - P_s(t))\gamma_s(t) + VP_s(t)\beta_s \\ & - Q_{g_s^*}(t) - Z_{g_s^*}(t)]\} \quad (24) \\ s.t. \quad & a_s(t) + b_s(t) + f_s(t) \geq \sum_{g:s_g=s} u_g(t^-) - \phi_s(t) \\ & \text{Constraints (8)(10)(11)(12).} \end{aligned}$$

Here, we show the solutions when $\beta_s > \gamma_s(t)$. The other case $\beta_s \leq \gamma_s(t)$ can be solved in a similar manner with $f_s(t) = 0$. Based on real cases of Amazon EC2 [4], the following inequality holds in general given the fact $P_s(t) \in [0, 1]$

$$[(1 - P_s(t))\gamma_s(t) + P_s(t)\beta_s] \leq \beta_s \leq \alpha_s.$$

The objective function of (24) is linear in $a_s(t)$, $b_s(t)$, and $f_s(t)$. There are four cases in terms of the workload queue length and the virtual queue length.

Case 1: $Q_{g_s^*}(t) + Z_{g_s^*}(t) \leq V(1 - P_s(t))\gamma_s(t) + VP_s(t)\beta_s$. The objective function is always non-negative. $a_s(t)$, $b_s(t)$, and $f_s(t)$ should be as small as possible.

Case 1.1: $\sum_{g:s_g=s} u_g(t^-) - \phi_s(t) \leq 0$. The reserved VMs remaining effective are sufficient to accommodate left-over jobs. Then we have $a_s(t) = b_s(t) = f_s(t) = 0$.

Case 1.2: $0 < \sum_{g:s_g=s} u_g(t^-) - \phi_s(t) \leq f_s^{max}$. Spot VMs should be launched to run left-over jobs with the reserved VMs remaining effective. Then we have $a_s(t) = b_s(t) = 0$ and $f_s(t) = \sum_{g:s_g=s} u_g(t^-) - \phi_s(t)$.

Case 1.3: $0 < \sum_{g:s_g=s} u_g(t^-) - \phi_s(t) - f_s^{max} \leq b_s^{max}$. On-demand and Spot VMs should be launched to run left-over jobs with the reserved VMs remaining effective. Then we have $a_s(t) = 0$, $b_s(t) = \sum_{g:s_g=s} u_g(t^-) - \phi_s(t) - f_s^{max}$, and $f_s(t) = f_s^{max}$.

Case 1.4: $0 < \sum_{g:s_g=s} u_g(t^-) - \phi_s(t) - f_s^{max} - b_s^{max} \leq a_s^{max}$. All three types of VMs should be acquired to accommodate left-over jobs. Then we have $a_s(t) = \sum_{g:s_g=s} u_g(t^-) - \phi_s(t) - f_s^{max} - b_s^{max}$, $b_s(t) = b_s^{max}$, and $f_s(t) = f_s^{max}$.

Case 2: $V(1-P_s(t))\gamma_s(t)+VP_s(t)\beta_s < Q_{g_s^*}(t)+Z_{g_s^*}(t) \leq V\beta_s$. **Case 3:** $V\beta_s < Q_{g_s^*}(t)+Z_{g_s^*}(t) \leq V\alpha_s$. **Case 4:** $V\alpha_s < Q_{g_s^*}(t) + Z_{g_s^*}(t)$. The solutions of these cases can be found in our technical report [22].

The intuition here is to trade the queueing delay for cost reduction by using the workload queue length and virtual queue length as a guidance for making job scheduling and VM purchasing decisions. Once $a_s(t)$, $b_s(t)$ and $f_s(t)$ are decided, the job scheduling decisions can be made based on Eqn. (22) and Eqn. (23).

In the standard Lyapunov optimization framework [20], decisions made at the current time slot do not have any influence on decision making at the subsequent time slots. This paper handles a more general case in which one scheduled job will occupy VM resources for m_g consecutive time slots, directly affecting the job scheduling and VM purchasing decisions in later times. We make the following design to our cost minimization algorithm, in order to achieve provable algorithmic optimality.

First, we group N time slots into a super time frame, where N is the fixed reservation period for reserved instances. Let m^{max} denote the maximum units of workload of all types of jobs. We assume that $N > m^{max}$. The above job scheduling and VM purchasing algorithm varies depending on which time slot it is running at : at a time slot $t \in [xN, (x + 1)N - m^{max}]$, where x could be any non-negative integer, the above algorithm remains the same; in a time slot $t \in [(x + 1)N - m^{max} + 1, (x + 1)N - 1]$, only jobs with $m_g \leq (x + 1)N - t$ are considered in the selection of g_s^*:

$$g_s^* = argmax_{g:g_s=s, m_g \leq (x+1)N-t}[Q_g(t) + Z_g(t)]. \quad (25)$$

This design indicates that a new type-g job is scheduled only if it can finish its service within the super time frame.

Second, we impose another constraint that instances reserved at t will remain effective in the time span $[t, N(\lfloor \frac{t}{N} \rfloor + 1) - 1]$ instead of $[t, t + N - 1]$. All instances reserved during one super time frame will be cleared out at the end of that super time frame. So within one super time frame, the ASP uses reserved VMs over time in a nondecreasing manner.

C. Performance Analysis

We next analyze the performance of the designed online algorithm in terms of queueing delay bound, no job dropping conditions and cost optimality.

Theorem 1 (Queueing Delay Bound): If $m_g D_g^{max} > \max\{m_g r_g^{max}, \epsilon_g\}$, then each workload queue $Q_g(t)$ and each virtual queue $Z_g(t)$ are upper bounded by $Q_g^{max} = V\sigma_g/m_g + m_g r_g^{max}$ and $Z_g^{max} = V\sigma_g/m_g + \epsilon_g$, respectively, $\forall t, \forall g \in [1, G]$. The SLA of each job can be guaranteed by $\frac{Q_g^{max}+Z_g^{max}}{\epsilon_g}$, $\forall g \in [1, G]$, if we set $\epsilon_g = \frac{Q_g^{max}+Z_g^{max}}{l_g}$.

Proof: Detailed proof is provided in [22].

Theorem 2 (No Job Dropping Conditions): There is no job dropping in each time slot if the two conditions are satisfied

$$a_{s_g}^{max} + b_{s_g}^{max} + f_{s_g}^{max} \geq (\sum_{g' \in G} m_{g'})(m^{max} r^{max} + \epsilon^{max}) \quad (26)$$

$$\frac{V\sigma_g}{m_g} \geq V\alpha^{max} + (\sum_{g' \in G} m_{g'})(m^{max} r^{max} + \epsilon^{max}), \forall g \in G. \quad (27)$$

Here, $\alpha^{max} = max\{\alpha_s, \forall s \in S\}$, $r^{max} = max\{r_g^{max}, \forall g \in G\}$, and $\epsilon^{max} = max\{\epsilon_g, \forall g \in G\}$.

Proof: With condition (26), we can prove $Q_g(t) + Z_g(t) \leq V\alpha^{max} + (\sum_{g' \in G} m_{g'})(m^{max} r^{max} + \epsilon^{max}), \forall g \in G$. Then condition (27) can guarantee $Q_g(t) + Z_g(t) \leq \frac{V\sigma_g}{m_g}$. According to the optimal solution of problem (19), there is no job dropping. Detailed proof is provided in [22].

We next prove the performance optimality of our online algorithm. Define χ as the vector of time-averaged arriving workload for different types of jobs, *i.e.*,

$$\chi_g = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} m_g r_g(t).$$

A workload arrival rate vector χ is said to be supportable if there exist job scheduling and VM purchasing algorithms with no job dropping and no violation of the SLA requirements, under which all workload queues can be stabilized. The set of all supportable vectors of workload arrival rates is defined as the capacity region C at the ASP. We call an algorithm $(1+\delta)$-optimal if the algorithm can support any χ such that $(1+\delta)\chi \in C$ for some $\delta > 0$.

Theorem 3 (Performance Optimality): Suppose conditions (26) and (27) are satisfied, and $\frac{(1+\delta)N}{N-m^{max}}\chi \in C$ for some $\delta > 0$, under our online algorithm we have :

$$\begin{aligned}
&\lim_{\kappa \to \infty} \frac{1}{\kappa N} \sum_{x=0}^{\kappa-1} \sum_{t=xN}^{(x+1)N-1} E[Cost(t)] \\
&\leq Cost^{\frac{(1+\delta)N}{N-m^{max}}} + \frac{B}{V} + \frac{(N-m^{max})(N-m^{max}-1)}{2VN} B_1 \\
&+ \frac{N-1}{2V} \sum_{g \in [1,G]} [(\epsilon_g)^2 + (m_g)^2 (r_g^{max})^2] \\
&+ \frac{m^{max}}{N} \sum_{s \in [1,S]} (\alpha_s a_s^{max} + \beta_s b_s^{max} + \beta_s f_s^{max}) \\
&+ \frac{(N-m^{max})(N-m^{max}-1)}{2N} \sum_{s \in [1,S]} (f_s^{max})(\beta_s - \gamma_s^{min}),
\end{aligned} \quad (28)$$

where B is given in Lemma 1, γ_s^{min} is the minimum spot price for type-s spot VMs, and $B_1 = \sum_{g \in [1,G]} [m_g r_g^{max} + 2u_g^{max} + \epsilon_g] u_g^{max}$. RHS is the $\frac{(1+\delta)N}{N-m^{max}}$-optimal cost plus a constant. Note that 1-optimal cost is the offline optimum for the cost-minimization problem in Eqn. (6).

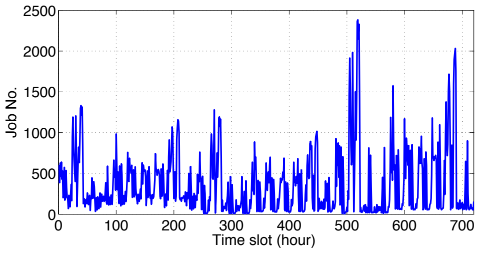

Fig. 3. The arrival pattern of one typical job type.

Proof: Detailed proof is provided in [22].

Theorem 1 and Theorem 3 show that, given a control parameter V, our algorithm is $O(1/V)$-optimal with respect to the average VM cost against the $\frac{(1+\delta)N}{N-m^{max}}$-optimal offline algorithm, while the queue length is bounded by $O(V)$. By increasing V, the developed online algorithm can push the time-average cost closer to the $\frac{(1+\delta)N}{N-m^{max}}$-optimal value at the expense of increasing the queueing delay at the ASP. Hence, by appropriately selecting the control parameter V, we can achieve a desired tradeoff between the VM cost and the queueing delay. If $V \to \infty$, $N \to \infty$ and $\frac{N}{V} < \infty$, our algorithm can achieve a time-averaged cost with a constant gap to the 1-optimal cost, *i.e.*, the offline optimum.

V. Performance Evaluation

In this section, we evaluate the performance of the proposed online algorithm through trace-driven simulations under realistic settings. We simulate an ASP which schedules and processes workloads with VMs purchased from one IaaS cloud provider like Amazon EC2 [2].

A. Simulation Setup

Job Types. We consider six types of VMs $S = \{$m3.xlarge, m3.2xlarge, c3.2xlarge, c3.4xlarge, r3.xlarge, r3.2xlarge$\}$. A job requiring some of the VMs lasts for different numbers of time slots. The number of time slots a job needs the VMs for, *i.e.*, the workload, is chosen randomly from $\{1, 2, 3, 4\}$. Hence, there are 24 types of jobs in total.

Demand Curve. We conduct our simulations based on Google cluster-usage traces [23], reflecting the resource demands (CPU, memory, etc.) of jobs submitted to the Google cluster. We translate the Google data into concrete hourly job arrival rates as our input. One time slot is an hour. Fig. 3 shows job arrivals at the ASP for one typical job type.

Pricing. The cost parameters in the problem formulation are all set according to Amazon EC2 pricing policies [4]. Specifically, the one-time payment for a reserved VM is calculated by scaling down the total charge (upfront fee plus hourly charges) of a real reserved instance with 1-year term according to the ratio $N/(365 * 24)$. The hourly price for running one on-demand VM is set as {\$0.28 \$0.56 \$0.42 \$0.84 \$0.35 \$0.7} for the six VM types, the same as the instances hosted on Amazon EC2. We extract real spot prices of the six VM types in the region US West (Oregon) of Amazon EC2 from Jun 22 to Jul 22, 2014 by Amazon EC2 CLI Tools [24].

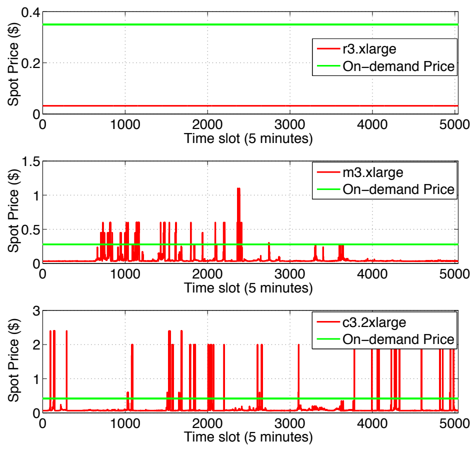

Fig. 4. The spot price fluctuations of three typical VM types.

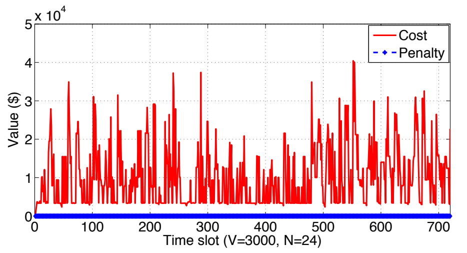

Fig. 5. Cost and penalty in each time slot.

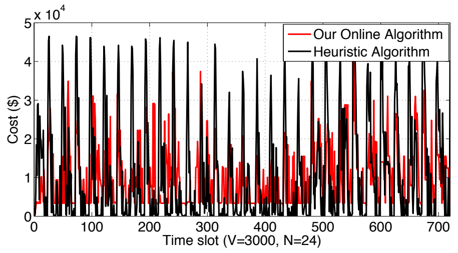

Fig. 6. Comparison of costs between our algorithm and heuristic algorithm.

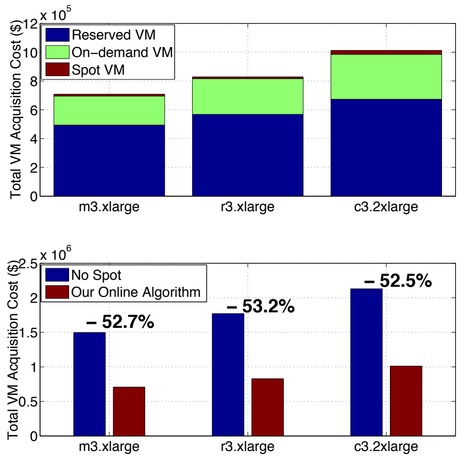

Fig. 7. Cost savings with and without spot VMs for different VM types.

Spot prices are updated every 5 minutes. Fig. 4 illustrates temporal variations in spot prices for three typical VM types, in contrast with fixed on-demand prices.

B. Cost

We first run our online algorithm for 720 time slots with parameters $V = 3000$, $N = 24$, $\epsilon_g = 50 * m_g$ and $\sigma_g = 1000 * \alpha_{s_g}$. We scale down the value of N due to the limit of available spot price traces, but it still represents the main characteristics of reserved instances. Fig. 5 shows the cost and penalty incurred by dropped jobs of the ASP in each time slot. We see that no penalty occurs under our setup, validating Theorem 2 in Sec. IV.

For comparison purposes, we also implement a heuristic algorithm that always maximally schedules type-$g_s^\dagger$ jobs, whose observed value of $Q_g(t)$ is the largest among all types of jobs requiring type-s VMs. Fig. 6 shows the costs achieved by the two algorithms respectively. We observe that our online algorithm outperforms the heuristic algorithm over time.

C. Impact of Spot Price Fluctuation

We illustrate the fractions of VM acquisition cost for three typical VM types with our online algorithm in the upper figure in Fig. 7. We observe that the VM acquisition cost comes primarily from reserved and on-demand VMs. To further understand the connection between spot price fluctuations and cost savings, we next evaluate the VM acquisition cost discount due to using and not using spot instances in our online algorithm for three typical VM types. The lower figure in Fig. 7 shows the cost advantage of our online algorithm over another online algorithm *No Spot* that conducts VM acquisition without spot VMs for comparison purposes. As illustrated in Fig. 4, spot prices for the three typical VM types present quite different temporal fluctuation patterns. But we see that our online algorithm can bring more than 50% cost savings for all three VM types, when spot instances are exploited. Therefore our algorithm is robust to the fluctuation of spot prices. We also observe that though the number of spot instances exploited by our online algorithm is small as compared to the numbers of reserved and on-demand instances, the cost saving is huge, due to the very low prices of spot instances.

D. Impact of V and N

We next study the scheduling delays experienced by jobs. Fig. 8 shows the average response delays and maximum response delays with our online algorithm under different values of V. We see that both average response delay and maximum response delay increase as V increases, confirming the impact of V on the queueing delay in Theorem 1.

From Theorem 3, we note that the cost performance of the proposed online algorithm depends on two critical factors, V and N. Fig. 9 reveals how the time-averaged cost achieved by our algorithm varies with different values of V and N. We see that as V increases, the time-averaged cost decreases, verifying

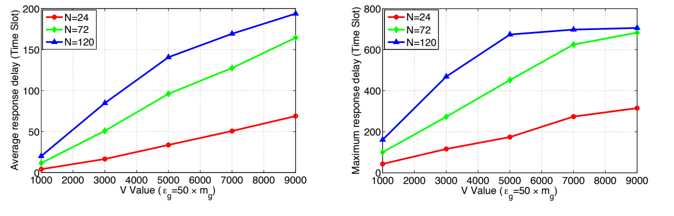
(a) (b)

Fig. 8. Average and maximum response delays under different values of V.

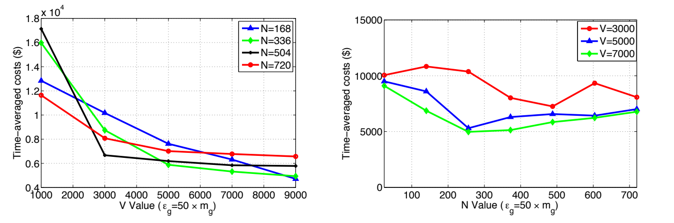
(a) (b)

Fig. 9. Time-averaged costs under different values of V and N.

the $O(1/V)$-versus-$O(V)$ cost-delay tradeoff in Theorem 1 and Theorem 3. N is the reservation period and the length of a super frame as well. Fig. 9 (b) suggests that the value of N has relatively less impact on the time-averaged cost of our online algorithm . When V increases, ϵ is properly set, and N is large enough, the time-averaged cost is arbitrarily close to the offline optimum plus a constant.

E. Characterizing Algorithm Robustness

As mentioned in Sec. IV, our algorithm needs to predict the probability of the occurrence of a termination event for type-s spot VMs in the coming time slot $P_s(t)$, which is estimated based on historical activity records of type-s spot VMs within a window of a certain number of past time slots. Now we explore the influence of the prediction window size on the cost performance. In Fig. 10, we show the time-averaged costs with different prediction window sizes. We can see that the cost performance of our online algorithm is completely insensitive to the prediction window size. Therefore, the proposed online algorithm is robust in our estimation.

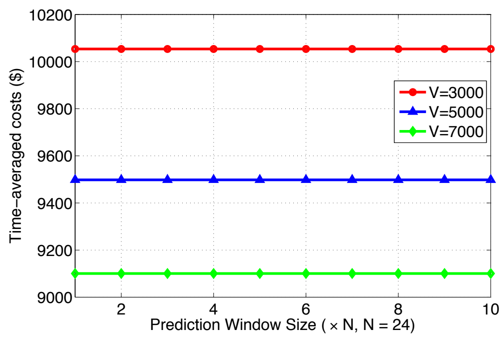
Fig. 10. Time-averaged costs with different prediction window sizes.

VI. Concluding Remarks

This paper investigates cost minimization strategies at ASPs with arbitrary demands which provision application services with VMs purchased from IaaS Clouds. We design a dynamic algorithm for an ASP to schedule job service/drop, and to decide the amount of reserved, on-demand and spot instances to purchase simultaneously in the most economic fashion, under time-varying job arrivals and fluctuating spot instance prices. The proposed algorithm can obtain a time-averaged VM purchasing cost with a constant gap from its offline minimum, based on solid theoretical analysis and trace-driven evaluations.

References

[1] https://www.netflix.com/global.
[2] http://aws.amazon.com/ec2/.
[3] http://aws.amazon.com/solutions/case-studies/netflix/.
[4] http://aws.amazon.com/ec2/pricing/.
[5] https://aws.amazon.com/premiumsupport/trustedadvisor/.
[6] H. Zhao, M. Pan, X. Liu, and Y. Fang, "Optimal resource rental planning for elastic applications in cloud market," in *Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2012.
[7] Y. Hong, J. Xue, and M. Thottethodi, "Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud," in *Proc. of ACM SIGMETRICS*, 2011.
[8] W. Wang, B. Li, and B. Liang, "To Reserve or Not to Reserve: Optimal Online Multi-Instance Acquisition in IaaS Clouds," in *Proc. of USENIX International Conference on Autonomic Computing*, 2013.
[9] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic Cloud Resource Reservation via Cloud Brokerage," in *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013.
[10] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data Centers Power Reduction: A Two Time Scale Approach for Delay Tolerant Workload," in *Proc. of IEEE INFOCOM*, 2012.
[11] S. Ren, Y. He, and F. Xu, "Provably-Efficient Job Scheduling for Energy and Fairness in Geographically Distributed Data Centers," in *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2012.
[12] U. Sharma, S. Prashant, S. Sahu, and A. Shaikh, "A Cost-Aware Elasticity Provisioning System for the Cloud," in *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011.
[13] X. Qiu, H. Li, C. Wu, Z. Li, and F. C. Lau, "Cost-Minimizing Dynamic Migration of Content Distribution Services into Hybrid Clouds," in *Proc. of IEEE INFOCOM, Mini Conference*, 2012.
[14] A. Khanafer, M. Kodialam, and K. P. N. Puttaswamy, "The Constrained Ski-Rental Problem and its Application to Online Cloud Cost Optimization," in *Proc. of IEEE INFOCOM*, 2013.
[15] V. Setty, R. Vitenberg, G. Kreitz, U. Guido, and M. v. Steen, "Cost-Effective Resource Allocation for Deploying Pub/Sub on Cloud," in *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2014.
[16] H. Roh, C. Jung, W. Lee, and D. Du, "Resource Pricing Game in Geo-distributed Clouds," in *Proc. of IEEE INFOCOM*, 2013.
[17] L. M. Leslie, Y. C. Lee, P. Lu, and A. Y. Zomaya, "Exploiting Performance and Cost Diversity in the Cloud," in *Proc. of IEEE International Conference on Cloud Computing*, 2013.
[18] S. Lu, X. Li, L. Wang, H. Kasim, H. Palit, T. Hung, E. F. T. Legara, and G. Lee, "A Dynamic Hybrid Resource Provisoning Approach for Running Large-scale Computional Applications on Cloud Spot and On-demand Instances," in *Proc. of IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2013.
[19] I. Menache, O. Shamir, and N. Jain, "On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud," in *Proc. of USENIX International Conference on Autonomic Computing*, 2014.
[20] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan&Claypool Publishers, 2010.
[21] https://aws.amazon.com/ec2/purchasing-options/reserved-instances/.
[22] http://i.cs.hku.hk/%7ecwu/papers/techreport-skshicloud15.pdf.
[23] http://code.google.com/p/googleclusterdata/.
[24] http://aws.amazon.com/documentation/cli/.