Towards Designing Cost-Optimal Policies to Utilize IaaS Clouds with Online Learning

Xiaohu Wu
Fondazione Bruno Kessler
Trento, Italy
xiaohuwu@fbk.eu

Patrick Loiseau
EURECOM
Sophia-Antipolis, France
patrick.loiseau@eurecom.fr

Esa Hyytiä
University of Iceland
Reykjavík, Iceland
esa@hi.is

***Abstract*—Many businesses possess a small infrastructure that they can use for their computing tasks, but also often buy extra computing resources from clouds. Cloud vendors such as Amazon EC2 offer two types of purchase options: on-demand and spot instances. As tenants have limited budgets to satisfy their computing needs, it is crucial for them to determine how to purchase different options and utilize them (in addition to possible self-owned instances) in a cost-effective manner while respecting their response-time targets. In this paper, we propose a framework to design policies to allocate self-owned, on-demand and spot instances to arriving jobs. In particular, we propose a near-optimal policy to determine the number of self-owned instance and an optimal policy to determine the number of on-demand instances to buy and the number of spot instances to bid for at each time unit. Our policies rely on a small number of parameters and we use an online learning technique to infer their optimal values. Through numerical simulations, we show the effectiveness of our proposed policies, in particular that they achieve a cost reduction of up to 62.85% when spot and on-demand instances are considered and of up to 44.00% when self-owned instances are considered, compared to previously proposed or intuitive policies.**

I. Introduction

Infrastructure as a Service (IaaS) holds exciting potential of elastically scaling users' computation capacity up and down to match their time-varying demand. This eliminates the users' need of purchasing servers to satisfy their peak demand, without causing an unacceptable latency. IaaS is seeing a fast growth and nowadays has become the second-largest public cloud subsegment [1], [2], accounting for almost half of all data center infrastructure shipments. Cost management in IaaS clouds is therefore a premier concern for users and has received significant attention.

Two common purchase options in the cloud are on-demand and spot instances. On-demand instances are always available with a fixed price and tenants[1] pay only for the period in which instances are consumed at an hourly rate. Furthermore, users can also bid a price for spot instances and can successfully get them only if their bid is above the spot price. Spot instances will then run as long as the bid is above the spot price but they will be terminated if the spot price becomes higher. Here, spot prices usually vary unpredictably over time and users will be charged the spot prices for their use [4]. Compared to on-demand instances, spot instances can reduce the cost by up to 50-90% [3].

Users purchasing computing instances on the cloud may have their own instances, referred to as self-owned instances, which can be used to process jobs but are insufficient at times (hence the need to purchase extra IaaS instances). They may also not have any self-owned instances (e.g., in the case of startups) and therefore need to buy from the cloud all necessary computing resources. In both cases though, the fundamental question for users is to determine how to purchase instances from IaaS clouds and utilize different instances to process their jobs in a way that minimizes their cost.

Tenants' jobs often have constraints that must be satisfied while trying to minimize cost. In particular, we consider here the classical constraints of parallelism bounds and deadlines [5], [6], [7], [8], [9]. The parallelism bound specifies the maximum number of instances that could be utilized by a job simultaneously and each job has to be completed by some deadline to satisfy its response-time requirement. Subject to the parallelism bound constraint, an arriving job will be allocated instances of different types (self-owned, on-demand and spot) and this allocation will be updated every hour (since billing is done per hour) until the job is completed. The problem is then to find the allocation that minimizes cost while ensuring that the job will be completed by its deadline.

Challenges. In this paper, we make the natural assumption that self-owned instances are cheaper than spot instances,[2] which are themselves cheaper than on-demand instances. Hence, to be cost-optimal, an allocation policy should allocate as many self-owned instances as possible, then spot instances, then on-demand instances. This is, however, a difficult task. For instance, a naive policy to achieve a high utilization of self-owned instances would be, when a job arrives, to assign as many remaining self-owned instances

[1] In this paper, we will use "users" and "tenants" interchangeably.

[2] Note that this assumes that the tenant has a sufficiently high load to justify maintaining the self-owned infrastructure on.

978-1-5386-1939-1/17 $31.00 © 2017 European Union
DOI 10.1109/ICCAC.2017.23

as possible to it. However, this policy turns out not to be good wrt cost. Indeed, it treats all jobs equally to assign self-owned instances, whereas a good policy wrt cost needs instead to assign more self-owned instances to jobs that will be less able to use spot instances (and hence have to use the more expensive on-demand instances) at times. Finding a policy that maximizes the opportunity to utilize spot instances while achieving a high utilization of self-owned instances is a challenging problem. Similarly, when allocating on-demand instances, one also needs to consider the capability of jobs to utilize spot instances in the future, in order to avoid that too many on-demand instances are consumed.

Our Contributions. In this paper, we propose a framework to design policies to allocate various instances. Based on the two principles that (i) self-owned instances should be allocated to maximize their utilization while maximizing the opportunity to utilize spot instances and (ii) on-demand instances should be allocated to maximize the opportunity to utilize spot instances, we propose parametric policies for the allocation of self-owned, on-demand and spot instances that achieve near-minimal costs. To cope with the cloud market dynamic, we use the online learning technique in [6], [7] to infer the optimal parameters. More specifically:

- We propose a cost-effective policy for the allocation of self-owned instances that is smarter than the naive allocation mentioned above and hits a good trade-off between utilization of self-owned instances and opportunity of utilization of spot instances, controlled by a parameter β_0. We show in our numerical experiments that this policy improves the cost by up to 44% compared to the naive policy.
- We propose a cost-optimal policy for the utilization of on-demand and spot instances, based on a formulation of the original problem as an integer program to maximize the utilization of spot instances. This policy can be used both when the tenant has self-owned resources and when he does not. Our simulation results show that it improves the cost of previous policies in [6], [7] by up to 62.85%.

The rest of this paper is organized as follows. We introduce the related works in Section II and describe the problem formally in Section III. In Section IV, we propose scheduling policies for self-owned, on-demand and spot instances, using online learning. In Section V, simulations are done to show the effectiveness of the solutions of this paper. Finally, we conclude this paper in Section VI.

II. Related Work

In this paper, we use online learning technique to learn the most-effective parameters for utilizing various instances[3]. Jain *et al.* were the first to consider the application of this approach to the scenario of cloud computing[4] [6], [7]. However, they do not consider the problem of how to optimally utilize the purchase options in IaaS clouds and self-owned instances are also not taken into account. The online learning approach is interesting because it does not impose the restriction of a priori statistical knowledge of workload, compared to other techniques such as stochastic programming (see Section IV-D for an introduction of online learning). However, it can achieve good performances only if optimal scheduling policies are proposed in the set of possible policies.

Similar to our paper and [6], [7], executing deadline-constrained jobs cost-effectively in IaaS clouds is also studied in [10], [11]. In particular, Zafer *et al.* characterize the evolution of spot prices by a Markov model and propose an optimal bidding strategy to utilize spot instances to complete a serial or parallel job by some deadline [10]. Yao *et al.* study the problem of utilizing reserved and on-demand instances to complete online batch jobs by their deadlines and formulate it as integer programming problems; then heuristic algorithms are proposed to give approximate solutions [11].

There have been substantial works on cost-effective resource provisioning in IaaS clouds [12]; in the following, we introduce some of the typical approaches used in this problem. There are many works with the assumption of a priori statistical knowledge of the workload or spot prices [13], [14], [22] and then several techniques could be applied. In [13], [14], the techniques of stochastic programming is applied to achieve the cost-optimal acquisition of reserved and on-demand instances. In [22], the optimal strategy for the users to bid for the spot instances are derived, given a predicted distribution over spot prices. However, when implementing these techniques, there is a high computation complexity although the statistical knowledge could be derived by the techniques such as dynamic programming [17].

Wang *et al.* use the competitive analysis technique to purchase reserved and on-demand instances without knowing the future workload [15], where the Bahncard problem is applied to propose a deterministic and a randomized algorithm. In [16], a genetic algorithm is proposed to quickly approximate the pareto-set of makespan and cost for a bag of tasks where on-demand and spot instances are considered. In [17], the technique of Lyapunov optimization is applied and it's said to be the first effort on jointly leveraging all three common IaaS cloud pricing options to comprehensively reduce the cost of users. The less interesting aspect of this technique is that a large delay will be caused when processing jobs; in order to achieve an $\mathcal{O}(\epsilon)$ close-to-optimal performance, the queue size has to be $\Theta(1/\epsilon)$ [18].

[3]We refer readers to [19] to see how such self-adaptive systems run in practice.

[4]The objective of this paper corresponds to a special case of [6], [7] where the value of each job is larger than the cost of completing it.

III. Problem Description and Model

In this section, we introduce the cloud pricing models, define the operational space of a user to utilize various instances, and characterize the objective of this paper.

A. Pricing Models in the Cloud

We first introduce the pricing models in the cloud. The price of an *on-demand instance* is charged on an hourly basis and it is fixed and denoted by p. Even if on-demand instances are consumed for part of an hour, the tenant will be charged the fee of the entire hour.

Tenants can also bid a price for *spot instances* and *spot prices are updated at regular time intervals* (e.g., every $L = 5$ minutes in Amazon) [22]. Spot instances are assigned to a job and continue running if the spot price is lower than the bid. Since spot prices usually change unpredictably over time [4], once the spot price exceeds the bid price of a job, its spot instances will get lost suddenly and terminated immediately by the cloud. The tenant will be charged the spot prices for the maximum integer hours of execution. A partial hour of execution is not charged in the case where its instances are terminated by the cloud; in contrast, if spot instances run until a job is completed and then are terminated by the tenant, for the partial hour of execution, the tenant will also be charged for the full hour.

In addition, a user can have its own computing instances, i.e., *self-owned instances*. The (averaged) hourly cost of utilizing self-owned instances is assumed to be p_1. We assume that it is the cheapest to use self-owned instances so that p_1 is without loss of generality assumed to be 0. An example of self-owned instances is academic private clouds, which are provided to researchers free of charge.

B. Jobs

The job arrival of a tenant is monitored every time slot of L minutes (i.e., at the time points when spot prices change) and time slots are indexed by $t = 1, 2, \cdots$. Each job j has four characteristics: (i) *an arrival slot* a_j: If job j arrives at a certain continuous time point in $[(t-1)L, tL)$, then set a_j to t; (ii) *a relative deadline* $d_j \in \mathcal{Z}^+$: every job must be completed at or before time slot $a_j + d_j - 1$; (iii) *a job size* z_j (measured in CPU time slots that need to be utilized); (iv) *a parallelism bound* δ_j: the upper bound on the number of instances that could be simultaneously utilized by j. The tenant plans to rent instances in IaaS clouds to process its jobs and aims to minimize the cost of completing a set of jobs $\mathcal{J}$ (that arrive over a time horizon T) by their deadlines.

C. General Rules for Allocating Resource to Jobs

The pricing models define the rules of allocating instances to jobs and also the operational space of a user, i.e., (a) when the resource allocation to jobs is done and updated, and, (b) how various instances and especially spot instances are utilized by jobs at every allocation update.

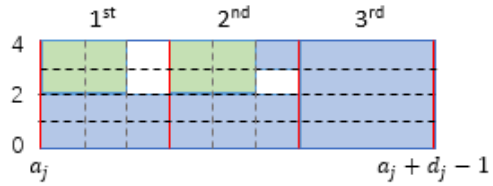

Figure 1. Illustration of the process of allocating resource to j where $\delta_j = 4$, d_j equals 3 hours, $L = 5$ minutes and $z_j = 132$: the area between two red lines illustrates the workload processed after an allocation update and the height of green (resp. blue) areas denote the number of spot (resp. on-demand) instances acquired from the cloud.

We first consider the allocation of on-demand and spot instances alone. Each job j is allocated instances to complete z_j workload by the deadline. To meet the deadline, we assume that (**i**) *whenever a job j arrives at a_j, the allocation of spot and on-demand instances to it is done immediately*.

The following rules are proposed for the case where there is the flexibility for j to utilize spot instances. Given the fact that the tenant is charged on hourly boundaries, (**ii**) *the allocation of on-demand and spot instances to each job j is updated simultaneously every hour.* In the i-th allocation, the number of on-demand instances allocated to j is denoted by o_j^i and they will be utilized for the entire hour.

At the i-th allocation of j, we assume that (**iii**) *the tenant will bid a price b_j^i for a fixed number si_j^i of spot instances.* At the i-th allocation of j, b_j^i together with the spot prices determines whether j can successfully obtain spot instances and for how long it can utilize them.

In this paper, we will apply an online learning approach. We do not assume exact statistical knowledge of spot prices over time; we only know that spot instances are on average cheaper than on-demand instances In such context, we assume that (**iv**) *at every allocation the tenant will bid for the maximum number of spot instances under the parallelism constraint, i.e., $si_j^i = \delta_j - o_j^i$.* The crucial question is therefore how to determine the proportion of on-demand and spot instances that are acquired from the cloud and allocated to j.

Before the i-th allocation of j, we denote by z_j^i the remaining workload of j to be processed, i.e., z_j minus the workload of j that has been processed, where $z_j^1 = z_j$, and we define the current slackness of j as

$$s_j^i = \frac{(d_j - (i-1) \cdot Len) \cdot \delta_j}{z_j^i}, \tag{1}$$

where $Len = \frac{60}{L}$ is the number of slots per hour. Let $s_j = s_j^1$. The slackness of a job can be used to measure the time flexibility that j has to utilize spot instances and the process of allocating on-demand and spot instances to j is in fact divided into two phases considering the deadline constraint:

Definition 1. *When spot instances are terminated by the cloud at the end of some slot t' and are not utilized for an*

entire hour at the i-th allocation update of j, we say that:

- *j has the flexibility to utilize spot instances at the next allocation update, if $s_j^{i+1} \geq 1$;*
- *j does not have such flexibility at the next allocation update, otherwise.*

We now illustrate Definition 1 by Fig. 1. As illustrated in Fig. 1, $z_j^1 = 132$ and, at the 1st allocation update, $o_j^1 = si_j^1 = 2$; then $z_j^2 = 132 - 2 \cdot 12 - 2 \cdot 8 = 92$. At the 2nd update, o_j^2 and si_j^2 are still 2 and then $z_j^3 = 92 - 2 \cdot 12 - 2 \cdot 8 = 52$. Further, $s_j^3 = \frac{Len \cdot \delta_j}{z_j^3} < 1$ and there is no flexibility for j to utilize unstable spot instances at the third allocation update. We use i_j to index the last allocation update after which there is no flexibility to utilize spot instances; in Fig. 1, $i_j = 2$.

When self-owned instances are taken into account, we assume that (**v**) *the allocation of self-owned instances to a job can be updated at most once at every allocation update of that job*. We denote by r_j^i the number of self-owned instances assigned to j at the i-th allocation. In this paper, o_j^i and si_j^i denotes the numbers of on-demand and spot instances acquired at the i-th update and will be used to track the cost of completing j. As we will see in Section IV-B, the acquired on-demand instances may not be fully utilized for an entire hour at the i_j-th allocation, and, we use $o_j(t)$, $si_j(t)$ and $r_j(t)$ to denote the numbers of on-demand, spot and self-owned instances that are actually utilized by j at every slot $t \in [a_j, a_j + d_j - 1]$, where $r_j(t) = r_j^i$ for all $t \in [a_j + (i-1) \cdot Len, a_j + i \cdot Len - 1]$. The parallelism constraint further translates to $o_j^i + si_j^i + r_j^i = \delta_j$ and $o_j(t) + si_j(t) + r_j(t) = \delta_j$.

In this paper, we apply the online learning approach and there is no exact statistical knowledge on the jobs and spot prices. At every allocation update of j, only the current characteristics of j (i.e., z_j^i, δ_j, a_j, and d_j) and the amount of available self-owned instances are definitely known for us to design scheduling policies. As a result, the scheduling policies can be the following function with a domain $\mathcal{Y}_j^i \times \mathcal{N}$, where $\mathcal{Y}_j^i$ is a set of the current characteristics of j and $\mathcal{N}$ is the amount of the current available self-owned instances.

Definition 2. *At the i-th allocation of j, the policy is a function $F : \mathcal{Y}_j^i \times \mathcal{N} \to (r_j^i, si_j^i, o_j^i)$, where $o_j^i + si_j^i + r_j^i = \delta_j$; the job j will be allocated r_j^i self-owned instances and o_j^i on-demand instances, and, bid some price for si_j^i spot instances.*

Furthermore, the value of spot prices is jointly determined by the arriving jobs of numerous users and the number of idle servers at a moment, usually varying over time unpredictably. In this paper, it is assumed that the change of spot prices over time is independent of the job's arrival of a user [10], [22]. At the i-th allocation update of j, when a user bids some price for si_j^i spot instances, without considering the case where the spot instances of j is terminated by a user itself, the expected time for which j could utilize spot instance is assumed to be $\beta \cdot Len$ where $\beta \in [0, 1]$.

D. Scheduling Objectives

We refer to the ratio of the total cost of utilizing a certain type of instances to the total workload processed by this type of instances as the average unit cost of this type of instances. As described in Section III-A, we assume that

Assumption 1. *The average unit costs of self-owned instances is lower than the average unit cost of spot instances, which is lower than that of on-demand instances.*

Accordingly, to be cost-optimal, we should consider allocating various instances to each arriving job in the order of self-owned, spot and on-demand instances. Further, in Principles 1 and 2, we give the objectives that should be achieved when considering allocating each type of instances to the arriving jobs.

Principle 1. *The scheduler should make self-owned instances (i) fully utilized, and (ii) utilized in a way so as to maximize the opportunity that all jobs have to utilize spot instances.*

Principle 2. *After self-owned instances are used, the scheduler should utilize on-demand instances in a way so as to maximize the opportunity that all jobs have to utilize spot instances.*

Principles 1 and 2 are intuitive under Assumption 1. A more formal analysis of them is relegated to our technical report [23] due to space limits.

In the subsequent optimization process of this paper, we consider how to propose policies for various instances so as to maximize the utilization of self-owned instances and further the utilization of spot instances. In other words, we will specify the function $F : \mathcal{Y}_j^i \times \mathcal{N} \to (r_j^i, si_j^i, o_j^i)$ in Definition 2 in a way such that Principles 1 and 2 are well realized, where only $\mathcal{Y}_j^i$ and $\mathcal{N}$ are known. Finally, Table I summarizes the main notation of this paper.

IV. The Design of Near-Optimal Policies

In this section, we propose a theoretical framework to design (near-)optimal parametric policies to realize Principles 1 and 2.

Facing diverse users, these policies have good adaptability against the uncertainties of spot prices, jobs' characteristics, and the amount of self-owned instances, that is, by applying the techniques such as online learning, the best configuration parameters could be inferred for each individual user to minimize its cost of processing jobs.

A. Self-owned Instances

Upon arrival of a job j, the scheduler first considers the allocation of self-owned instances to it with the aim to realize the two goals in Principle 1.

Table I
MAIN NOTATION

Symbol	Explanation
L	length of a time slot (e.g., 5 minutes)
Len	the number of time slots in an hour, i.e., $\frac{60}{L}$
$\mathcal{J}$	a set of jobs that arrive over time
j and a_j	a job of $\mathcal{J}$ and its arrival time
d_j	the relative deadline: j must be completed by a deadline $a_j + d_j - 1$
z_j	the job size of j, measured in CPU $\times$ time slots
δ_j	the parallelism bound, i.e., the maximum number of instances that can be simultaneously used by j
s_j	the slackness, i.e., $\frac{d_j}{z_j/\delta_j}$ where z_j/δ_j denotes the minimum execution time of j
T	the number of time slots, i.e., $\max_{j\in\mathcal{J}}\{a_j\}$
si_j^i, b_j^i, and o_j^i	the number of spot instances bid for, the bid price, and the number of on-demand instances acquired at the i-th allocation update of j
$r_j(t)$, $si_j(t)$ and $o_j(t)$	the number of self-owned, spot and on-demand instances utilized by j at a slot t
p_j^i	the spot price charged at the i-th allocation of j
z_j^i	the remaining workload of j to be processed at the i-th allocation update to j
s_j^i	the slackness at the i-th allocation update, i.e., $(d_j - (i-1)\cdot Len)\cdot\delta_j/z_j^i$
p and p_1	the price of respectively using an on-demand and self-owned instance for an hour
R	the number of self-owned instances
$\{\beta, \beta_0, b\}$	a tuple of parameters that defines a policy and determines the allocation of various instances to j at every allocation
$\mathcal{P}$	a set of parameterized policies, each indexed by π and defined by $\{\beta, \beta_0, b\}$
r_j	the number of self-owned instances allocated to a job j at every $t \in [a_j, a_j + d_j - 1]$

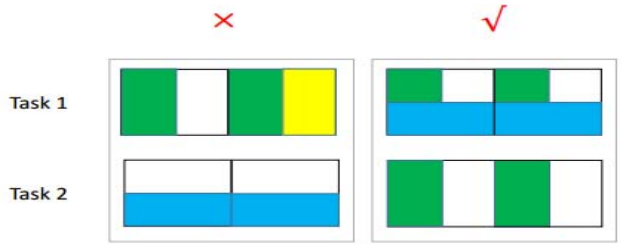

Figure 2. The Challenge in Cost-Effectively Utilizing Self-owned Instances.

Challenges. We first show the challenges in cost-effectively utilizing self-owned instances by an example. Let $N(t)$ denote the number of self-owned instances that are currently idle at a slot t; let $m_{t_1}(t_2) = \min\{N(t_1), \cdots, N(t_2)\}$, where $t_1 \le t_2$, and $m_{t_1}(t_2)$ represents the maximum number of self-owned instances idle at every slot in $[t_1, t_2]$. An intuitive policy would be, whenever a job j arrives, to allocate as many self-owned instances to j to make self-owned instances fully utilized, i.e.,

$$r_j = \min\{m_{a_j}(a_j + d_j - 1), z_j/d_j\}. \tag{2}$$

However, this intuitive policy may not maximize the opportunity that all jobs have to utilize spot instances as illustrated in the following example.

Consider the statitc case illustrated in Fig. 2 where there are two jobs and a self-owned instance available. These two jobs have the same arrival time, relative deadline of 2 hours and parallelism bound of 4. The first and second jobs respectively have a size of $6\times Len$ and $4\times Len$. Assume that it is expected that a job can utilize β hour ($\beta\cdot Len$ slots) of spot instances at every allocation update where $\beta = \frac{1}{2}$. As illustrated in Fig. 2, the green, blue and yellow areas denote the workload respectively processed by spot, self-owned and on-demand instances. In the first allocation, the user has to purchase two on-demand instances for one hour while it is not necessary to purchase the more expensive on-demand instances, as illustrated by the second allocation. One can observe that the second allocation is also cost-optimal.

To sum up, we want the policy for self-owned instances to have the following adaptability to realize Principle 1. In the case where there are limited instances, a cost-effective policy should have the ability (i) to choose the subset of all jobs (that are expected to be completed by totally utilizing spot instances without on-demand instances) such that they will not be allocated any self-owned instances, and (ii) not to allocate too many self-owned instances to the rest of jobs such that it is expected that the job's remaining workload either could not be completed or is exactly completed (e.g., the right allocation to the first task in Fig. 2) by the deadline by totally utilizing spot instances. In the case where there are adequate instances, the policy should have the ability to ensure that after the allocation of self-owned instances to each job, it could be completed by the deadline by totally utilizing spot instances. At the same time, in both cases, the policy should ensure that self-owned instances are fully utilized.

Policy Design. In the following, we propose a policy that has the abilities described above. In the subsequent analysis, the issue of rounding the allocations of a job to integers is ignored temporarily for simplicity; in reality, we could round the allocations up to integers, which does not affect the related conclusions much as shown by the analysis.

Let $\kappa_0 = \lceil\frac{d_j}{Len}\rceil - 1$,

$$r'_j(\beta) = \delta_j - \frac{d_j\cdot\delta_j - z_j}{d_j - (\kappa_0+1)\cdot Len\cdot\beta}, \tag{3}$$

and,

$$r''_j(\beta) = \delta_j - \frac{d_j\cdot\delta_j - z_j}{(1-\beta)\cdot\kappa_0\cdot Len}. \tag{4}$$

Let $\overline{r}_j(\beta)$ equal $r'_j(\beta)$ in the case where $d_j - \kappa_0\cdot Len \ge \beta\cdot Len$ and equal $r''_j(\beta)$ in the case where $d_j - \kappa_0\cdot Len < \beta\cdot Len$.

Proposition 1. *A job can be expected to be completed by utilizing only spot instances after the allocation of self-owned instances if and only if* r_j^i *is set to* $\max\{\overline{r}_j(\beta), 0\}$.

Proof: We analyze the two cases. The first one is $d_j - \kappa_0 \cdot Len \geq \beta \cdot Len$. In this case, if a job can be expected to be completed by the deadline by totally utilizing spot instances after the allocation of self-owned instances, it could be expected that

$$r_j \cdot d_j + (\kappa_0 + 1) \cdot (\delta_j - r_j) \cdot Len \cdot \beta \geq z_j.$$

This makes us derive that $r_j \geq r'_j(\beta)$. The second case is $d_j - \kappa_0 \cdot Len < \beta \cdot Len$. In this case, it is expected that

$$\begin{aligned} r_j \cdot d_j + \kappa_0 \cdot (\delta_j - r_j) \cdot Len \cdot \beta \\ + (d_j - \kappa_0 \cdot Len) \cdot (\delta_j - r_j) \geq z_j. \end{aligned}$$

This makes us derive that $r_j \geq r''_j(\beta)$. As a summary of our analysis of both cases, the proposition holds. ■

As a corollary of Proposition 1, we also clarify that

Proposition 2. *Given a job j, if $\overline{r}_j \leq 0$, it can be expected that, without allocation of self-owned instances, j can be completed by the deadline by utilizing only spot instances.*

Based on Proposition 1, upon arrival of a job j, we propose the following policy for allocating self-owned instances:

$$r_j(\beta_0) = \min\left\{\max\left\{\overline{r}_j(\beta_0), 0\right\}, m_t(a_j + d_j - 1)\right\}, \quad (5)$$

where $\beta_0 \in [0, 1)$ is a parameter and r_j^i is a non-increasing function of β_0 since $z_j/d_j - \delta_j \leq 0$.

This policy achieves the more cost-effective resource allocation as illustrated in the second allocation in Fig. 2 by setting $\beta_0 = \beta$. What's more, the policy defined in (5) is also adaptive. When a user owns more self-owned instances (e.g., 4 instances), β_0 can be set to be smaller than β (e.g., 0); then, both jobs will be allocated two self-owned instances. As a result, self-owned instances are fully utilized and there is no need purchasing spot or on-demand instances.

Now, we further explain the reason why the policy defined by (5) could well realize Principle 1, which is also validated by the simulations.

Given a set of jobs $\mathcal{T}$ that arrive over time, let $\mathcal{T}'_\beta$ denote all jobs j with $\overline{r}_j(\beta) \leq 0$, and $\mathcal{T}''_\beta = \mathcal{T} - \mathcal{T}'_\beta$. The jobs of $\mathcal{T}'_\beta$ could be expected to be completed by the deadline by totally utilizing spot instances even without being allocated any self-owned instances. In contrast, all jobs of $j \in \mathcal{T}''_\beta$ are expected to have to utilize some of more expensive on-demand instances to be completed by the deadlines when allocated less than $\overline{r}_j(\beta)$ self-owned instances.

The online learning technique used subsequently in Section IV-D has the capacity to learn the most cost-effective β_0. In terms of $\mathcal{T}$, when there are less self-owned instances, we can set β_0 to a value that satisfies the following two conditions to realize Principle 1: (i) set β_0 to a value no less than β, which leads to that

1) no self-owned instances will be allocated to any job in $\mathcal{T}'_\beta$, and,
2) all jobs of $\mathcal{T}''_\beta$ are allocated less than $\max\{\overline{r}_j(\beta), 0\}$ self-owned instances,

and, in the meantime, (ii) control the value of β_0 not to be too large, which can ensure that

- every job of $\mathcal{T}''_\beta$ is allocated an properly large number of self-owned instances to make self-owned instances fully utilized, which realizes one goal of Principle 1.

In the following, we show that the allocation of self-owned instances to $\mathcal{T}'_\beta$ and $\mathcal{T}''_\beta$ does not affect the capacity that each job $j \in \mathcal{T}$ has to utilize spot instances much.

Even if no self-owned instances is allocated to a job $j \in \mathcal{T}'_\beta$, all its workload could be completed by totally utilizing spot instances; in this case, if some instances were allocated to j, less self-owned instances would be available by $\mathcal{T}''_\beta$ whose jobs have to utilize some amount of self-owned or on-demand instances to be completed by the deadlines, so that, they have to utilize more on-demand instances.

Once a job $j \in \mathcal{T}''_\beta$ is allocated r_j self-owned instances, its remaining workload to be processed could be viewed as a new job j' that will processed by utilizing spot and on-demand instances alone with the same arrival time and deadline as j but a reduced size $z_j - r_j \cdot d_j$ and parallelism bound $\delta_j - r_j$. As we will see in Proposition 5, with the expected optimal strategy to utilize spot and on-demand instances, the maximum workload of a job that could be processed by spot instances mainly depends on the product of its relative deadline and parallelism bound minus its total workload to be processed. Hence, after the allocation of self-owned instances to $j \in \mathcal{T}''_\beta$, its capacity to utilize spot instances is almost not harmed.

When there are adequate self-owned instances, setting β_0 to a value no less than β will lead to a low utilization of self-owned instances and a waste of them, and, we could set β_0 to a properly small value that is less than β. Then, each job will consume enough self-owned instances to achieve a high utilization of self-owned instances, and, the amount of self-owned instances allocated to each job is also no less than $\max\{\overline{r}_j(\beta), 0\}$, which leads to that it is expected that the remaining workload of each job is completed by totally utilizing spot instances. Hence, Principle 2 is also realized.

B. Spot and On-demand Instances

Once a job j is allocated r_j self-owned instances at all $t \in [a_j, a_j + d_j - 1]$, it can be viewed as a new job where spot and on-demand instances alone are utilized. So, without loss of generality, we are to consider the case with on-demand and spot instances alone to simplify the analysis.

Now, we analyze the expected cost-optimal policy when there is the flexibility for j to utilize unstable spot instances. Let $\kappa_1 = \lceil \frac{t' - a_j + 1}{Len} \rceil$, representing the total number of allocation updates at which it has the opportunity to utilize spot instances. Here, recall that t' is first given in Definition 1. After the i-th allocation update of j where $1 \leq i \leq \kappa_1$,

it is expected that the workloads processed by spot and on-demand instances are respectively $(\delta_j - o_j^i) \cdot Len \cdot \beta$ and $o_j^i \cdot Len$. As indicated by Definition 1, j has the last opportunity to utilize unstable spot instances at the κ_1-th allocation update, i.e.,

$$s_j^{\kappa_1} = \frac{z_j - \sum_{i=1}^{\kappa_1-1} o_j^i \cdot Len - \sum_{i=1}^{\kappa_1-1} (\delta_j - o_j^i) \cdot Len \cdot \beta}{\delta_j (d_j - (\kappa_1 - 1) \cdot Len)} \geq 1,$$

and has no opportunity to utilize spot instances at the $(\kappa_1 + 1)$-th allocation update, i.e.,

$$s_j^{\kappa_1+1} = \frac{z_j - \sum_{i=1}^{\kappa_1} o_j^i \cdot Len - \sum_{i=1}^{\kappa_1} (\delta_j - o_j^i) \cdot Len \cdot \beta}{\delta_j \cdot (d_j - \kappa_1 \cdot Len)} < 1.$$

This corresponds to the following relations:

$$\sum_{i=1}^{\kappa_1-1} (\delta_j - o_j^i) \cdot Len \cdot (1-\beta) \leq d_j \cdot \delta_j - z_j \quad (6)$$

$$\sum_{i=1}^{\kappa_1} (\delta_j - o_j^i) \cdot Len \cdot (1-\beta) > d_j \cdot \delta_j - z_j \quad (7)$$

In this subsection, our objective is to maximize the total workload processed by spot instances at the first κ_1 allocations, i.e.,

$$\text{maximize} \ \sum_{i=1}^{\kappa_1} (\delta_j - o_j^i) \cdot Len \cdot \beta, \quad (8)$$

subject to the constraints (6) and (7).

Now, we give the optimal solution to (8). Given any feasible solution of (8), we can increase the value of $o_j^{\kappa_1}$ to the maximum value δ_j and this increases our objective function (8) without leading to the violation of constraints (6) and (7). Hence, in the optimal solution, the value of the term $\delta_j - o_j^{\kappa_1}$ is δ_j. Further, the optimal solution of (8) is obtained when the term $\sum_{i=1}^{\kappa_1-1} (\delta_j - o_j^i)$, i.e., the total number of spot instances bid for at the first $\kappa_1 - 1$ allocation updates, equals

$$\nu(z_j, d_j) = \left\lfloor \frac{d_j \cdot \delta_j - z_j}{Len \cdot (1-\beta)} \right\rfloor. \quad (9)$$

Let $\kappa_2(z_j, d_j) = \lfloor \frac{\nu(z_j,d_j)}{\delta_j} \rfloor$. We also illustrate the above optimal solution to (8) by Fig. 3 where the orange and green areas denote the workload processed respectively by spot and on-demand instances; in the grey areas, no workload of j is processed. We assume that $\beta = \frac{1}{2}$ and $L = 5$; as a result $L = 12$. The job j has $d_j = 42$ (3.5 hours), $z_j = 122$ and $\delta_j = 4$. From the left to the right, the first four subfigures illustrate the expected optimal allocation of spot and on-demand instances. Here, we have $\nu(z_j, d_j) = 7$ and $\kappa_2(z_j, d_j) = 1$. Hence, an optimal allocation of spot and on-demand instances is as follows. At the first $\kappa_2(z_j, d_j)$ allocation updates of j, $\delta_j = 4$ spot instances are bid for and the expected execution time of spot instances is $\beta \cdot Len = 6$. At the $(\kappa_2(z_j, d_j) + 1)$-th allocation update, $(\nu(z_j, d_j) - \delta_j \cdot \kappa_2(z_j, d_j))$ spot instances are bid for and one on-demand instance is purchased for an hour. So far, $\nu(z_j, d_j) = 7$ spot instances have been bid for. At the $(\kappa_2(z_j, d_j) + 2)$-th allocation update, δ_j spot instances are bid for and after the execution of spot instances, j has no opportunity to utilize spot instances and it turns to totally utilize on-demand instances as illustrated by the fourth subfigure.

To better understand the optimality of the above allocation, in contrast, we also use the last three subfigures to illustrate an intuitive way to bid for spot instances where δ_j spot instances are bid for at every allocation update when j has the flexibility to utilize spot instances. We should notice that at the second allocation update, if δ_j spot instances are bid for, the expected remaining workload of j after utilizing spot instances of the second allocation update could not be completed if j is not allocated any on-demand instance until at the third allocation update. It is worth noting that in the optimal allocation, the expected maximum workload processed by spot instances is $(\nu(z_j, d_j) + \delta_j) \cdot Len \cdot \beta$; given the value of β, it only depends on the job's characteristics of δ_j and $\delta_j \cdot d_j - z_j$.

Finally, once we obtain the optimal solution to (8), as illustrated by Fig. 3, we can conclude that

Proposition 3. *To maximize the total workload processed by spot instances, an expected optimal strategy is to **(i)** bid for δ_j spot instances at the first $\kappa_2(z_j, d_j)$ allocation updates, and, **(ii.a)** in the case where $\kappa_2(z_j, d_j) \cdot \delta_j = \nu(z_j, d_j)$,*

- *bid for δ_j spot instances at the $(\kappa_2(z_j, d_j) + 1)$-th allocation update where $\kappa_1 = \kappa_2(z_j, d_j) + 1$,*

***(ii.b)** in the case where $\kappa_2(z_j, d_j) \cdot \delta_j < \nu(z_j, d_j)$,*

- *bid for $\nu(z_j, d_j) - \kappa_2(z_j, d_j) \cdot \delta_j$ spot instances at the $(\kappa_2(z_j, d_j)+1)$-th allocation update and bid for δ_j spot instances at the $(\kappa_2(z_j, d_j) + 2)$-th allocation update where $\kappa_1 = \kappa_2(z_j, d_j) + 2$.*

Based on Proposition 3, we propose Algorithm 1 to dynamically determine how many on-demand and spot instances are allocated to j at each of its allocation updates in the case where j has the flexibility to utilize spot instances.

In the following, we analyze the optimal utilization of on-demand instances in the case where a job j has no flexibility to utilize spot instances. Let $\kappa = \lfloor \frac{a_j + d_j - 1 - t'}{Len} \rfloor$, denoting the maximum integer multiple of an hour (containing Len slots) in $[t' + 1, a_j + d_j - 1]$, and $t'' = a_j + d_j - \kappa \cdot Len$ where $0 < t'' - 1 - t' < Len$. Given the fact that on-demand instances are charged on an hourly basis, we conclude that

Proposition 4. *Once spot instances are terminated by the cloud at the i_j-th allocation, in the case where j has no flexibility to utilize spot instances at the next allocation update,*

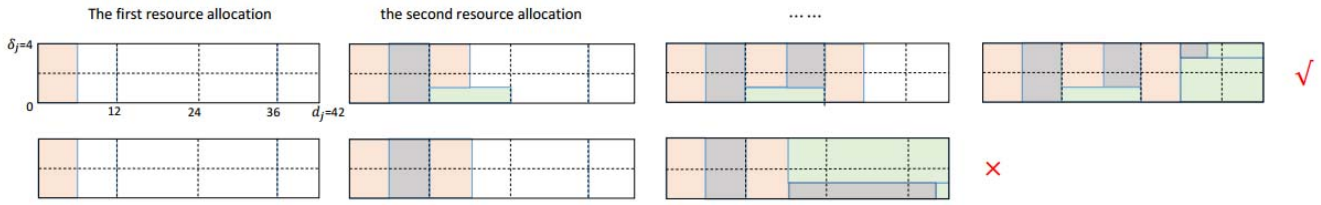

Figure 3. From the left to the right, the first four figures (top) denote the optimal allocation of spot instances; in comparison, the last three figures (bottom) denote the allocation of spot instances where δ_j spot instances are bid for at every allocation update when j has the flexibility to utilize spot instances.

Algorithm 1: Proportion(j, β, b)

/* At the current allocation update of j, the remaining workload of j to be processed could be viewed as a new job with the arrival time t, workload z'_j, parallelism bound δ_j, and relative deadline $a_j + d_j - t$ */

if $\kappa_2(z'_j, a_j + d_j - t) \geq 1$ **then**
  $si_j^i \leftarrow \delta_j$, $o_j^i \leftarrow 0$;
if $\kappa_2(z'_j, a_j + d_j - t) = 0 \wedge \nu(z_j, a_j + d_j - t) > 0$ **then**
  $si_j^i \leftarrow \nu(z_j, a_j + d_j - t)$, $o_j^i \leftarrow \delta_j - si_j^i$;
if $\nu(z_j, a_j + d_j - t) = 0$ **then**
  $si_j^i \leftarrow \delta_j$, $o_j^i \leftarrow 0$;
$b_j^i \leftarrow b$;
at the i-th allocation update, bid a price b_j^i for si_j^i spot instances;

Algorithm 2: Dynalloc

Input : at a slot t, a job j to be processed with the characteristics $\{a_j, d_j, z'_j, \delta_j\}$ and a parameterized policy $\{\beta_0, \beta, b\}$

if $a_j = t$ **then**
  // upon arrival of j, allocate self-owned instances to it
  set the value of r_j using Equation (5);
  for $\bar{t} \leftarrow a_j$ **to** $a_j + d_j - 1$ **do**
    $r_j(\bar{t}) \leftarrow r_j$;
$i \leftarrow \left\lfloor \frac{t - a_j}{Len} \right\rfloor + 1$;
if $\frac{t-a_j}{Len} = i - 1$ **then**
  // at the i-th allocation update of j, j has the flexibility for spot instances
  call Algorithm 1;
if *the spot instances of j are terminated at $t-1$* **then**
  if $\frac{(\delta_j - r_j)\cdot(d_j - Len\cdot i)}{z'_j} < 1$ **then**
    // j has no flexibility to utilize spot instances at the next allocation update
    apply the strategy in Proposition 4 here;
  // otherwise, j still has the flexibility at the next allocation update where $z'_j = z_j^{i+1}$

the cost-optimal strategy to utilize on-demand instances is to

- *acquire δ_j on-demand instances to be utilized at every slot $t \in [t'', a_j + d_j - 1]$;*
- *acquire $\bar{o}$ more on-demand instances to be utilized at every $t \in [t' + 1, t'' - 1]$, where $\bar{o} = (z_j^{i_j+1} - \kappa \cdot \delta_j \cdot Len)/(t'' - t')$.*

Proof: Due to the limitation of pages, we refer readers to [23] for the detailed proof. ■

To sum up, in Propositions 3 and 4, we have given the expected optimal strategy for numerous jobs to utilize spot and on-demand instances. Finally, we give the maximum workload processed by spot instances using the expected optimal policy in Proposition 3, which is also given by the optimal solution to (8):

Proposition 5. *Given a job j, the expected maximum workload processed by spot instances is*

$$(\nu(z_j, d_j) + \delta_j) \cdot Len \cdot \beta.$$

Proposition 5 shows that, the expected maximum workload of a job processed by spot instances mainly depends on the product of its relative deadline and parallelism bound minus its total workload to be processed (i.e., $\nu(z_j, d_j)$ defined in (9)), given the value of β. This also helps us understand that, after the allocation of self-owned instances, our policy in Section IV-A will not much harm the capacity that the remaining workload of a job has to utilize spot instances, in order to be completed by the deadline.

C. Scheduling Framework

As described above, a general policy is defined by a tuple $\{\beta_0, \beta, b\}$ and determines the amounts of self-owned, spot, and, on-demand instances allocated to a job, and, the bid price, also specifying the function F in Definition 2.

The complete framework used to determine the allocation of self-owned, spot and on-demand instances to every such j at t is presented as Algorithm 2 where z'_j denotes the

remaining workload of j to be processed after deducting its current allocation from the total workload of j. At the beginning of every slot t, for every job j that is not completed at this moment, it may arrive at or before the slot t, and, the algorithm will check the state of j to decide how to allocate computing instances to it.

D. The Application of Online Learning

In this subsection, we show how online learning is applied to learn the most cost-effective parameters $\{\beta_0, \beta, b\}$. The online learning algorithm that we adopt is the one in [6], [7], presented as Algorithm 3, and is also a form of the classic weighted majority algorithm.

It runs as follows. There are a set of jobs $\mathcal{J}$ that arrive over time and a set of scheduling policies $\mathcal{P}$ each specified by $\{\beta_0, \beta, b\}$ in Section IV-C. Let $d = \max_{j\in\mathcal{J}}\{d_j\}$, i.e., the maximum relative deadline of all jobs. Let $\mathcal{J}_t \subseteq \mathcal{J}$ denote all jobs, each of which j arrives at the time slot t, i.e., $a_j = t$. There is also an initial distribution over n policies, e.g., a discrete uniform distribution $\{1/n, \cdots, 1/n\}$.

Whenever a job $j \in \mathcal{J}_t$ arrives, the algorithm randomly picks a policy from $\mathcal{P}$ according to the distribution and bases the allocation of various instances to j on that policy. In the meantime, when $t > d$, if $\mathcal{J}_{t-d} \neq \emptyset$, since the history of spot prices in the time interval $[a_j - d, a_j - 1]$ has been known, we are enabled to compute the cost of each policy on a job in $\mathcal{J}_{t-d}$. Subsequently, the weight of each policy (i.e., its probability) are updated so that the lower-cost (higher-cost) polices of this job are re-assigned the enlarged (resp. reduced) weights. As more and more jobs are processed and the above process repeats, the most cost-effective policies of $\mathcal{P}$ will be identified gradually, i.e., the ones with the highest weights, well realizing Principles 1 and 2 and finally minimizing the total cost of completing all jobs.

As modeled in Section III, the cost of completing a job is from the use of spot and on-demand instances alone and is defined as their cost. For each job $j \in \mathcal{J}$, let π_j denote the policy selected by Algorithm 2 under which j is completed. Denote by $c_j(\pi)$ the cost of completing a job j under some policy $\pi \in \mathcal{P}$. Let $N' = |\cup_{t=d+1}^{T} \mathcal{J}_t|$, i.e., the number of all jobs that arrive in $[d+1, T]$, and, as proved in [7], we have that

Proposition 6. *For all $\delta \in (0, 1)$, it holds with a probability at least $1 - \delta$ over the random of online learning that*

$$\max_{\pi\in\mathcal{P}} \left\{ \sum_{t\in\cup_{t=d+1}^{T}\mathcal{J}_t} \frac{c_j(\pi_j) - c_j(\pi)}{N'} \right\} \leq 9\sqrt{\frac{2d\log(n/\delta)}{N'}}.$$

Proposition 6 says that, as an online learning algorithm runs, the actual total cost of completing all jobs is close to the cost of completing all jobs under a policy $\pi^* \in \mathcal{P}$ that generates the lowest total cost. Recall that a policy is defined by a tuple of parameters from $\mathcal{P}$.

Algorithm 3: OptiLearning

Input : a set $\mathcal{P}$ of n policies, each π being parameterized so that $\pi \in \{1, 2, \cdots, n\}$; the set $\mathcal{J}_t$ of jobs that arrive at t;

initialize the weight vector of policies of dimension n: $w_1 = \{w_{1,1}, \cdots, w_{1,n}\} = \{1/n, \cdots, 1/n\}$;
for $t \leftarrow 1$ **to** T **do**
  if $\mathcal{J}_t \neq \emptyset$ **then**
    for each $j \in \mathcal{J}_t$, pick a policy π with a probability $w_{j,\pi}$, being applied to j;
  if $t \leq d$ **then**
    $w_{j+1} \leftarrow w_j$;
  else
    while $\mathcal{J}_{t-d} \neq \emptyset$ **do**
      $\eta_t \leftarrow \sqrt{\frac{2\log n}{d(t-d)}}$;
      get a job j from $\mathcal{J}_{t-d}$;
      for $\pi \leftarrow 1$ **to** n **do**
        $w'_{j+1,\pi} \leftarrow w_{j,\pi} \exp^{-\eta_j c_j(\pi)}$;
      for $\pi \leftarrow 1$ **to** n **do**
        $w_{j+1,\pi} \leftarrow \frac{w'_{j+1,\pi}}{\sum_{i=1}^{n} w'_{j+1,i}}$;
      $\mathcal{J}_{t-d} \leftarrow \mathcal{J}_{t-d} - \{j\}$;

V. Evaluation

The main aim of our evaluations is to show the effectiveness of the proposed policies of this paper.

A. Simulation Setups

The on-demand price is $p = 0.25$ per hour. We set L to 5 (minutes) and all the jobs in the experiments have a parallelism bound of 20. Following [20], [21], we generate the jobs as follows. The job's arrival is generated according to a poisson distribution with a mean of 1. The size z_j of every job j is set to $12 \times 20 \times x$ where x follows a bounded Pareto distribution with a shape parameter $\epsilon = \frac{1}{1.01}$, a scale parameter $\sigma = \frac{1}{6.06}$ and a location parameter $\mu = \frac{1}{6}$; the maximum and minimum value of x is set to 0.5 and 10. The job's relative deadline is generated as $x \cdot z_j/\delta_j$, where x is uniformly distributed over $[1, x_0]$. x represents the slackness of a job and, as shown by our analysis and especially by Propositions 1 and 5, is a main factor that determines the performance. Spot prices are updated every time slot and their values can follow an exponential distribution where its mean is set to 1.1 [22].

As shown in Section IV, the policies are defined by the parameters β, β_0 and b. Both β and β_1 are chosen from $\{\frac{i}{12} | 0 \leq i \leq 11\} \cup \{0.9999\}$. The bid price b is chosen in $\{b_i = 0.1 + 0.03 \cdot (i-1) | 1 \leq i \leq 7\}$. In addition, in [6], [7], the algorithm will randomly select a parameter

$\theta \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ for every job j; $\theta \cdot \delta_j$ spot instances and $(1-\theta) \cdot \delta_j$ on-demand instances are acquired from the cloud for processing j. The main performance metric used to evaluate the policies of this paper is:

- *the average unit cost of each policy*, i.e., the total cost of utilizing instances to the total workload processed by them when this policy is applied to processed jobs, denoted by α.

B. Results for the Optimal Cases

In this subsection, we evaluate two types of jobs respectively with a small $x_0 = 3$ and a large $x_0 = 13$. This leads to two types of jobs with an expected slackness of 2 and 7 respectively.

When a user possesses x_1 self-owned instances, under the x_2-th type of jobs, we use α_{x_1,x_2} (resp. α'_{x_1,x_2}) to denote the minimum of the average unit costs of our policies (resp. the policies in [6], [7] and defined by (2)), where $x_2 = 1$ or 2. In this subsection, we mainly compute $\rho_{x_1,x_2} = 1 - \alpha_{x_1,x_2}/\alpha'_{x_1,x_2}$, that represents how much the performance of the policies defined in [6], [7] or by (2) is improved by when replaced by our policies.

In addition, we also show how the parameters such as bid prices and the number of self-owned instances are determining the best β and β_0 that lead to the minimum cost of completing all jobs, coherent with our related analysis in Section IV.

Experiment 1. This experiment aims to evaluate the proposed policies in Algorithm 1 and Proposition 4 for spot and on-demand instances without considering self-owned instances.

The simulation results are illustrated in Fig. 4. The green (resp. magenta) stars and circles respectively represent the average unit costs of our policies and the policies in [6], [7] under the first (resp. second) type of jobs. In terms of the best performance of policies, Table II shows a noticeable cost reduction when replacing the policies in [6], [7] with our policies, up to 62.85%.

As far as the policies proposed in this paper are concerned, starting from the first policy in Fig. 4, every 13 policies are divided into the same group where the i-th group uses the bid price b_i and $b_i < b_{i+1}$ for all $1 \le i \le 6$. In the same group, the parameter β in the 1st, 2nd, $\cdots$, and 13th policies are successively set to $\frac{0}{12}$, $\frac{1}{12}$, $\cdots$, $\frac{11}{12}$, 0.9999. Theoretically, as analyzed in Proposition 3, the expected time $\beta \cdot Len$, for which a job could utilize spot instances at each of its allocation updates, determines the cost-optimal policy and, in turn, is determined by the bid price. Each group of policies uses the same bid price and the minimum average unit cost is achieved when β is set to the right value.

As a result, under different types of jobs, in each group, the policies that achieve the minimum average unit cost has the same value of β. Furthermore, in a group that uses a higher bid price, a larger β is expected when the minimum average unit cost is achieved. In the simulation results illustrated in Fig. 4, for each group of policies, we use the red circle to mark the policy that generates the minimum average unit cost. The simulation results are coherent with our theoretical analysis.

$\rho_{0,1}$	$\rho_{0,3}$
52.70%	62.85%

Table II
PERFORMANCE IMPROVEMENTS

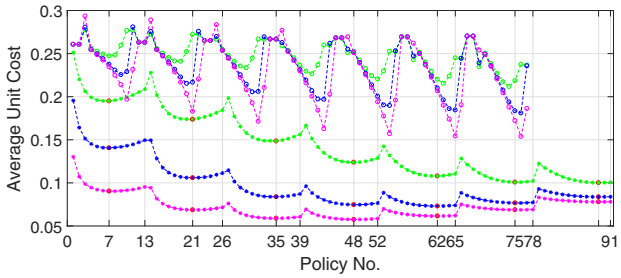

Figure 4. The average unit cost of each policy.

Experiment 2. This experiment aims to evaluate the proposed policy for self-owned instances.

For each group of 13 policies, the minimum average unit cost is also achieved by the same policy. Since the right value for β is determined by the bid price as shown in our analysis in Section IV-B and simulation results, we can compute the optimal β^* in advance and there is finally a total of 7 policies with different bid prices.

As a result, with different bid prices and β_0, 91 of our policies defined by (5) and 7 policies defined by (2) are to be evaluated for comparison. We take the simulations under the first and second type of jobs respectively with 200, 400, 600 and 800 self-owned instances and Table III also shows a noticeable cost reduction when replacing the policies defined by (2) with our policies, up to 44.00%.

We also illustrate the average unit costs of policies under the first type of jobs in Fig. 5 (left) and the corresponding utilizations of self-owned instances in Fig. 5 (right) where the magenta, blue, red and green stars represent the simulation results for the cases respectively with 200, 400, 600, and 800 self-owned instances. As far as the policies proposed in this paper are concerned, starting from the first policy in Fig. 5 (left), every 13 policies are divided into the same group where the i-th group uses the bid price b_i and $b_i < b_{i+1}$ for all $1 \le i \le 6$. In the same group, the parameter β_0 in the 1st, 2nd, $\cdots$, and 13th policies are successively set to $\frac{0}{12}$, $\frac{1}{12}$, $\cdots$, $\frac{11}{12}$, 0.9999, which is also the case of the simulation results in Fig. 5 (right).

From Fig. 5 (left), we can observe that, under the same bid price, the minimum average unit cost is achieved with

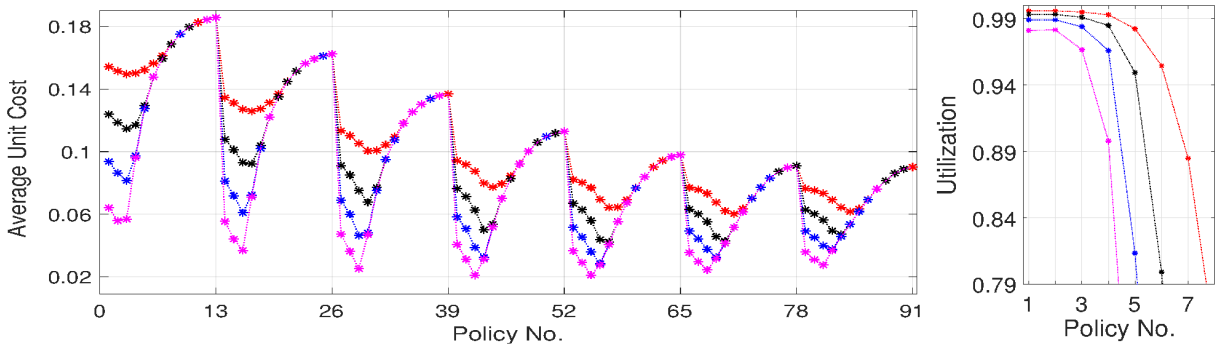

Figure 5. The average unit cost of each policy and the utilization of self-owned instances under different β_0.

	ρ_{200,x_2}	ρ_{400,x_2}	ρ_{600,x_2}	ρ_{800,x_2}
$x_2 = 1$	18.65%	28.01%	34.87%	31.68%
$x_2 = 3$	36.22%	44.00%	38.22%	20.28%

Table III
PERFORMANCE IMPROVEMENT TO SELF-OWNED INSTANCES

$\overline{\rho}_0$	$\overline{\rho}_{200}$	$\overline{\rho}_{400}$	$\overline{\rho}_{600}$	$\overline{\rho}_{800}$
59.56%	62.34%	56.04%	61.60%	61.66%

Table IV
PERFORMANCE IMPROVEMENT TO SELF-OWNED INSTANCES

a smaller β_0 if there are more self-owned instances. This is due to that more self-owned instances need to be consumed by the arriving jobs. A larger bid price will lead to a larger time for which a job could utilize spot instances at each of its allocation updates. Given the number of self-owned instances, with a larger bid price, the minimum average unit cost is achieved when β_0 is set to a larger value. This is coherent with our analysis in Section IV-A when we explain the reason why the policy defined by (5) could better realize Principle 1, compared with the intuitive policy defined by (2).

From Fig. 5 (right), the utilization of self-owned instances under the policy defined by (2) is a little higher than the utilization of the best policy of ours under which the minimum average unit cost is achieved. Even so, our best policy can achieve a markedly reduction in the cost of completing all jobs compared with the policy of (2), as shown in Table III. Given a set of jobs that arrive over time, a higher utilization of self-owned instances means more workload of jobs processed by them and less workload that remains to be processed by spot and on-demand instances. Even if more workload of jobs is processed by self-owned instances under the policy of (2), our policy can reduce the average unit cost of completing all jobs by up to 44.00%, as shown by Table III. Such cost reduction shows that our policy could greatly maximize the capacity that all jobs have to utilizing cheaper spot instances, realizing Principle 1 much better.

C. Results for the Online Learning Case

In this subsection, we show the cost of completing all jobs when online learning is applied.

We generate the third types of jobs in the same way as the first and second types except that x_0 is set to 5. When only on-demand and spot instances are considered, the average unit costs of our policies (resp. the policies of [6], [7]) under the third type of jobs are also illustrated in Fig. 4 with blue stars (circles). Here, like the case in Experiment 1 where the first and second types of jobs are considered, in each group of 13 policies that use the same bid price, the minimum average unit cost under the third type of jobs is achieved by the same policy.

As discussed in Experiments 1 and 2, in terms of the policies proposed in this paper, there are a total of 7 policies to be evaluated when only on-demand and spot instances are considered and a total of 91 policies when self-owned instances are also taken into account. Then, we run Algorithm 3 over about 30000 jobs respectively in the case where a user possesses 0, 200, 400, 600 and 800 computing instances.

When there are x_1 self-owned instances, under the third type of jobs, we use $\overline{\alpha}_{x_1}$ (resp. $\overline{\alpha}'_{x_1}$) to denote the cost of completing all jobs with our policies (resp. the policies in [6], [7] and defined by (2)). We define $\overline{\rho}_{x_1} = 1 - \overline{\alpha}_{x_1}/\overline{\alpha}'_{x_1}$, and, $\overline{\rho}_{x_1}$ represents how much the performance of the policies in [6], [7] and defined by (2) is improved by when replaced by our policies. The simulation results are given in Table IV and show that the proposed policies could reduce the cost noticeably, compared with the ones defined by [6], [7] and (2).

VI. CONCLUDING REMARK

Utilizing IaaS clouds cost-effectively is an important concern for all users. In this paper, we consider the problem of how to utilize different purchase options including spot and on-demand instances, in addition to possibly existing self-owned instances, to minimize the cost of processing all

incoming jobs while respecting their response-time targets. Driven by the goal of maximizing the utilization of self-owned instances while optimizing the possibility of utilizing spot instances, we propose policies for the allocation of these three types of instances that achieve small costs. These policies use online learning to infer the optimal values of their parameters. Through numerical simulations, we show the effectiveness of our proposed policies, in particular that they achieve a cost reduction of up to 62.85% when spot and on-demand instances are considered and of up to 44.00% when self-owned instances are considered.

Note that, in our paper, we have not considered the possibility that if a job allocated to a spot instance finishes before the end of the hour, the spot instance could be re-allocated to another job for the rest of the hour rather than being terminated by the tenant. That could possibly reduce the cost further although it would significantly complicate the allocation.

ACKNOWLEDGMENT

Part of Xiaohu Wu's work was done when he was with Eurecom, France and Aalto University, Finland. Esa Hyytia's work and a part of Xiaohu's work were supported by the Academy of Finland in the FQ4BD project (grant no. 296206). Patrick Loiseau acknowledges support from the Alexander von Humboldt Foundation.

REFERENCES

[1] "Cisco Maintains Lead in Public Cloud Infrastructure while HP Leads in Private." https://www.srgresearch.com/articles/cisco-maintains-lead-public-cloud-infrastructure-while-hp-leads-private.

[2] "Sizing the Public Cloud Computing Market." http://softwarestrategiesblog.com/2011/06/01/sizing-the-public-cloud/.

[3] Featured AWS case study. https://aws.amazon.com/ec2/purchasing-options/.

[4] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, Dan Tsafrir. "Deconstructing Amazon EC2 Spot Instance Pricing." *ACM Transactions on Economics and Computation*, 2013.

[5] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. "Jockey: Guaranteed Job Latency in Data Parallel Clusters." In *ACM EuroSys*, 2012.

[6] Navendu Jain, Ishai Menache, Ohad Shamir. "Allocation of Computational Resources with Policy Selection." U.S. Patent 20,130,246,208, Issued September 19, 2013.

[7] Ishai Menache, Ohad Shamir, Navendu Jain. On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud. In *USENIX ICAC*, 2014.

[8] Navendu Jain, Ishai Menache, Joseph Naor, and Jonathan Yaniv. "Near-Optimal Scheduling Mechanisms for Deadline-Sensitive Jobs in Large Computing Clusters." *ACM Transactions on Parallel Computing*, 2015.

[9] Xiaohu Wu, and Patrick Loiseau. "Algorithms for Scheduling Deadline-Sensitive Malleable Tasks." In *IEEE Allerton*, 2015.

[10] Murtaza Zafer, Yang Song, and Kang-Won Lee. "Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs." In *IEEE CLOUD*, 2012.

[11] Min Yao, Peng Zhang, Yin Li, Jie Hu, Chuang Lin, and Xiang-Yang Li. "Cutting Your Cloud Computing Cost for Deadline-Constrained Batch Jobs." In *IEEE ICWS*, 2014.

[12] Sunilkumar S. Manvi and Gopal Krishna Shyam. "Resource Management for Infrastructure as a Service (IaaS) in Cloud Computing: A Survey." *Journal of Network and Computer Applications (Elsevier)*, 2014.

[13] Yu-Ju Hong, Jiachen Xue, and Mithuna Thottethodi. "Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud." In *ACM SIGMETRICS*, 2011.

[14] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. "Optimization of Resource Provisioning Cost in Cloud Computing." *IEEE Transactions on Services Computing*, 2012.

[15] Wei Wang, Baochun Li, and Ben Liang. "Optimal Online Multi-Instance Acquisition in IaaS Clouds." *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[16] Alexandra Vintila, Ana-Maria Oprescu, and Thilo Kielmann. "Fast (Re-) Configuration of Mixed On-demand and Spot Instance Pools for High-Throughput Computing." In *ACM Workshop on Optimization Techniques for Resources Management in Clouds*, 2013.

[17] Shengkai Shi, Chuan Wu, and Zongpeng Li. "Cost-Minimizing Online VM Purchasing for Application Service Providers with Arbitrary Demands." In *IEEE CLOUD*, 2015.

[18] Longbo Huang, Xin Liu, and Xiaohong Hao. "The Power of Online Learning in Stochastic Network Optimization." In *ACM SIGMETRICS*, 2014.

[19] Nikolaus Huber, Jurgen Walter, Manuel Bahr, and Samuel Kounev. "Model-Based Autonomic and Performance-Aware System Adaptation in Heterogeneous Resource Environments: A Case Study." In *IEEE ICCAC*, 2015.

[20] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. "Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud." In *ACM HPDC*, 2011.

[21] Liang Zheng, Carlee Joe-Wong, Christopher G. Brinton, Chee Wei Tan, Sangtae Ha, and Mung Chiang. "On the Viability of a Cloud Virtual Service Provider." In *ACM SIGMETRICS*, 2016.

[22] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, and Xinyu Wang. "How to Bid the Cloud." In *ACM SIGCOMM*, 2015.

[23] Xiaohu Wu, Patrick Loiseau, and Esa Hyytiä. "Towards Designing Cost-Optimal Policies to Utilize IaaS Clouds under Online Learning." *arXiv:1607.05178v5 (Preprint)*, 2017.