An Analytical and Extended Cost-Effective Resource Provisioning Framework in IaaS Clouds under Online Learning

Xiaohu Wu  
Aalto University, Finland  
xiaohu.wu@aalto.fi

Patrick Loiseau  
EURECOM, France  
patrick.loiseau@eurecom.fr

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

Abstract—Cloud vendors such as Amazon EC2 offer three types of purchase options: reserved, on-demand and spot instances. An important problem for all users is determining the way of utilizing all purchase options to minimize the cost of processing all jobs while respecting the response-time targets. The application of online learning to this scenario is interesting in that it imposes no restriction of a priori statistical knowledge of workload and spot prices and achieves a good performance close to that of the best policy of the used set. So far, this approach can only address spot and on-demand instances and we are interested in enabling utilizing all purchase options and self-owned instances with it, which brings to users the opportunity to further reduce their cost. What’s more, we also lay some mathematical foundation for taking a holistic view to analyze and design a set of cost-optimal or effective policies that determine how many self-owned, spot and on-demand instances are assigned to each job and the final cost of completing all jobs. Finally, simulations are done, showing a markedly cost reductions when different combinations of self-owned, reserved, spot, and on-demand instances are considered.

I. INTRODUCTION

A. Background

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

In particular, there are three purchase models in the cloud [1]: on-demand instances, reserved, and spot instances. On-demand instances are always available with a fixed price and tenants pay only for the period in which instances are consumed on an hourly rate. Alternatively, tenants can also access reserved instances and are billed for every hour during the entire prescribed term that the tenant selects, regardless of whether the instance is running or not. The term is typically fixed and long (e.g., 1 or 3 years). Furthermore, tenants can also bid a price for spot instances and can successfully get them only if their bid price is above the spot price; spot instances will run until the bid price is below the spot price. Here, spot prices usually vary unpredictably over time and tenants will be charged the spot prices for their use [4]. Compared to on-demand instances, reserved instances can have a significant discount up to 75% and spot instances reduce the cost by up to 50-90% [3].

B. Motivations and Results

In this paper, we are interested in the use of online learning for cost-optimally resource provisioning from IaaS clouds. This is appealing in that online learning both does not impose the restriction of a priori statistical knowledge of workload, compared with other techniques such as stochastic programming (see the related works in Section I-C for more explanation), and it achieves a good performance if effective scheduling policies can proposed. Here, a policy determines how many of various cloud instances with different prices are utilized for completing a user’s jobs. So far, online learning has been applied to cloud resource provisioning but in the case where only on-demand and spot instances are present [5], [6]. The model used is as follows. A user’s jobs arrive over time and each job \( j \) is characterized by an arrival time \( a_j \), a deadline \( d_j \), a workload \( z_j \), and a parallelism bound \( \delta_j \); the parallelism bound \( \delta_j \) determines the maximum number of instances that the job can utilize simultaneously, and each job \( j \) must be completed by the time \( a_j + d_j \). In this paper, we consider the objective of minimizing the cost of utilizing cloud instances to complete the jobs of a tenant by their deadlines [5].

Facing both the user’s workload dynamics and the cloud instances with attractive features in different aspects, we aim to provide a holistic view of how to utilize self-owned, spot and on-demand instances cost-effectively, extending the previous online learning approach to the cases incorporating self-owned and reserved instances. In the following, we explain the challenges in cost-effectively utilizing cloud instances and introduce our solutions.

1. On-demand and Spot Instances. As far as on-demand and spot instances are concerned alone, since on-demand instances

---

\(^2\)This objective corresponds to a special case in [5] where the value of each job is larger than the cost of completing it.
are charged on an hourly basis, a rule of utilizing on-demand and spot instances is to update the allocation of them to each job \(j\) every hour and at each update \(i\), the scheduler should determine how many spot instances are bid for, denoted by \(s_i^j\), and how many on-demand instances are purchased, denoted by \(o_i^j\), where \(o_i^j + s_i^j = \delta_j\). Let \(f_i^j = s_i^j / \delta_j \in [0, 1]\). Here, the factor determines the cost include (i) the parameter \(f_i^j\) that determines \(s_i^j\) and \(o_i^j\), and (ii) the price \(b_i^j\) that a user bids for spot instances at each allocation update of \(j\). The basic idea of online learning is as follows. Given two finite sets \(\mathcal{P}_1\) and \(\mathcal{P}_2\) and let \(\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2\) denote a set of parameterized policies where a tuple of parameters \((f, b) \in \mathcal{P}\) determines the whole process in which on-demand and spot instances are utilized by \(j\) once the rule of allocating resource to \(j\) is defined, and online learning takes the responsibility to learn the most cost-effective tuple \((f', b') \in \mathcal{P}\). After numerous jobs are processed, the performance of online learning is close to the cost using the best tuple \((f', b') \in \mathcal{P}\). The above second factor that determines the cost has been extensively studied so far \([13, 14, 15]\) and this paper concerns the first factor.

At the \(i\)-th allocation (update) of \(j\), we use \(z_i^j\) to denote the remaining workload to be processed and \(d_i^j\) to denote the remaining time in which \(j\) has to be completed with the deadline constraint where \(z_i^j = z_j\) and \(d_i^j = d_j\). Let \(s_i^j = \frac{d_i^j - d_j}{\delta_j}\), referred to as the slackness of \(j\) at the \(i\)-th allocation update, which is used to represent the flexibility or opportunity that \(j\) has to utilize unstable spot instances. For example, consider a job \(j\) with \(\delta_j = 2\) and \(z_j = 2\) (in CPU hours). If \(d_j = 1\) hour, the allocation to \(j\) can be updated once and \(s_1^j = 1\); then, \(j\) can only bid for spot instances once. If \(d_j = 2\) hour, the allocation to \(j\) can be updated twice and \(s_2^j = 2\); then, \(j\) can bid for spot instances twice.

**Result.** We assume that utilizing spot instances is cheaper than on-demand instances on average after processing numerous jobs. Since spot prices change unpredictably over time, a common used assumption is that the change of spot prices over time is independent of the job arrival of a user \([15]\). Using this assumption, in spite of the complexity of mathematical analysis, the optimal policy that we derive is surprisingly simple: for every job \(j\), the scheduler bids for \(\delta_j\) spot instances at every allocation update until \(j\) has to totally utilize safe on-demand instances to meet the deadline. A simulation is done to validate the optimality of this principle.

Due to the diversity of users, sometimes, the above assumption on the job’s arrival and spot prices may does not hold and a new challenge arises when the following event happens: at the early several allocation updates of \(j\), the spot prices vary sharply and can become very high in a short time, while in the later allocation updates of \(j\), the spot prices vary in a range of lower prices most of the times. As a result, each job can only utilize a very short period of spot instances at the earlier allocation updates and, with the slackness decreasing, have to totally utilize on-demand instances to meet the deadline, thus reducing the opportunity of utilizing spot instances. To overcome this challenge, we propose a family of parameterized policies and online learning can learn the best value such that only if the slackness of \(j\) is below this value, \(j\) will always totally utilize on-demand instances to increase its slackness at every allocation update (see Section III-C for details). Such parameterized policies can keep most of the jobs with a relatively large flexibility to utilize spot instances. Under our simulation, we show that the performance of our policy reduces the average cost by 32.82%, compared with the policy in \([5]\). In practice, for many users, the generation of their jobs (e.g., production jobs) is roughly periodical or follows some statistical feature \([16, 17]\). Here, a user does not need to know the exact statistical feature of job’s arrival and it can empirically observe the pattern of generating jobs and the history of spot prices and judge whether that assumption holds, thus determining which policy to adopt.

2. **Self-owned Instances.** The second concern is how to cost-effectively utilize self-owned instances (i.e., the servers possessed by a user itself). Then, online learning can be extended to the case incorporating self-owned instances directly.

**Result.** We assume that using self-owned instances is always the cheapest and self-owned, spot and on-demand instances should be utilized in the increasing order of their prices. To minimize the cost, an ideal policy should achieve two goals while utilizing self-owned instances: (i) a high utilization, and (ii) maximizing the opportunity that all jobs have to utilize spot instances. However, the following challenge arises when design policies: a policy that achieves the first objective (e.g., whenever a job arrives, assign as many the remaining self-owned instances as possible to it) may deviate from the second objective. The reason for this is that such a policy processes all jobs identically and, given a limited number of self-owned instances, the jobs that already have large enough flexibilities may consume too many self-owned instances; in the case that there is a significant portion of jobs with small flexibilities, a good policy should assign more self-owned instances to a job with a smaller flexibility. To overcome this challenge, we propose a family of parameterized policies and, with them, online learning can learn the best value such that (i) in the case where the slackness of a job \(j\) is above this value, no self-owned instances will be allocated to it and (ii) in the case where its slackness is below this value, the smaller its slackness is, the more self-owned instances will be allocated to it. A simulation shows that, even though a small amount of self-owned instances are available, the average cost will be markedly lower than the cost without self-owned instances by 23.53%.

3. **Reserved Instances.** The third concern of this paper is how to apply the online learning approach to the case also incorporating reserved instances. Reserved instances have a long-term commitment (e.g., 1 or 3 years) and they are not worth purchasing if the workload of a user is not stable \([11]\). The online learning algorithm runs as follows. Let \(\mathcal{P}\) denote a set of policies and an initial distribution over \(\mathcal{P}\) (e.g., uniform distribution). Whenever a job \(j\) arrives, although a particular policy is selected randomly from \(\mathcal{P}\) for \(j\), the cost
of completing \( j \) under every policy will be computed and the policies achieving lower (resp. higher) costs will be reassigned enlarged (resp. reduced) probabilities; as a consequence of repeatedly executing the above process, the policies with the lowest (resp. higher) cost will be finally assigned the highest probabilities (probabilities close to 0). As stated in [5], an interesting direction for future research is incorporating reserved instances, and, the actions of purchasing reserved instances make the algorithm stateful and affect the payoffs (costs) of policies chosen in the future; this does not accord with the current framework of online learning.

**Result.** Here, the infeasibility to incorporate reserved instances holds only in the case where the workload varies unpredictably over time and a user would repeatedly consider purchasing reserved instances. In many scenarios such as production jobs, the generation of them is periodical or they follow some stochastic feature [5], [16] of which we don’t need have the exact knowledge here. In such cases, it is advisable to consider purchasing reserved instances since we could estimate the utilization of reserved instances in the long run. Accordingly, based on our result for self-owned instances, we propose a solution to the case incorporating reserved instances. The simulation shows that purchasing reserved instances can further reduce a user’s cost by 14.89%.

### C. Related Work

Now, we introduce the related approaches used in cost management in IaaS clouds. So far, many works have been done with the assumption of a priori knowledge of the workload or accurate prediction of future workload [7], [8], [9], [13]. In [7], [8], the techniques of stochastic programming is applied to achieve the cost-optimal acquisition of reserved and on-demand instances. In [9], the proposed algorithm iteratively makes decisions of purchasing reserved and on-demand instances based on the workload predictions. In [13], the optimal strategy for the users to bid for the spot instances are derived, given a predicted distribution over spot prices. However, these approaches suffer from high computation complexity although the statistical knowledge may be obtained by the techniques such as dynamic programming, and are suitable for online decision making in practice. Wang et al. use the competitive analysis technique to purchase reserved and on-demand instances without knowing the future workload [10], where the Bahnocard problem is applied to propose a deterministic and a randomized algorithm. In [5], [6], the online learning approach is applied to purchase on-demand and spot instances. In [11], 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 and in order to achieve an \( O(\epsilon) \) close-to-optimal performance, the queue size has to be \( \Theta(1/\epsilon) \) [12].

**Roadmap.** This paper is organized as follows. In Section [11] we describe the problem of this paper and lay some mathematical foundation for us to analyze what scheduling policies are cost-effective for various instances. In Section III, we propose scheduling policies for self-owned, on-demand and spot instances. In Section IV, the online learning approach is applied to our scheduling framework in Section III, addressing the cases incorporating self-owned and reserved instances. In Section V, simulations are done to show the effectiveness of the solutions of this paper. Finally, we conclude in Section [VI].

## II. 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.

**On-demand Instances.** The price of an on-demand instance is charged on an hourly basis and it is fixed and denoted by \( p \). Even if partial hour of on-demand instances is consumed, the tenant will be charged the fee of the entire hour.

**Spot Instances.** Tenants can bid a price for spot instances and spot prices are updated at regular time intervals (e.g., every \( L = 5 \) minutes in Amazon) [13]. They are assigned to a job and run until the spot price exceeds the bid price; 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 and the partial hour of execution is not charged, if its instances are terminated by the cloud; however, 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.

**Long-term Instances.** Reserved instances and self-owned instances are called long-term instances. Reserved instances can only be reserved for a fixed long term (e.g., 1 or 3 years) and tenants have to pay for these instances in every hour of that term even if they are not being used. The (average) hourly costs of reserved and self-owned instances are assumed to be \( p_2 \) and \( p_1 \), where \( p_1 < p_2 < p \). We assume that it is always the cheapest to use self-owned instances and the cost \( p_1 \) is without loss of generality assumed to be 0. A good example of self-owned instances be 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 \( d_j \in \mathbb{Z}^+ \); 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 \( \delta_j \); 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

In this subsection, based on the pricing models, we propose the rules of allocating instances to jobs. These rules define when the resource allocation to jobs is done and updated and how various instances and especially spot instances are utilized by jobs at every allocation update.

Firstly, we 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 instances to it is done immediately. The follow 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 done immediately upon its arrival and updated simultaneously every hour. In the first allocation, the number of on-demand instances allocated to \( j \) is assumed to be \( o^0_j \) and they will be utilized for the entire hour. If any of the spot instances is terminated by the tenant during the execution, the entire hour of fee will be charged by the cloud. At the \( i \)-th allocation of \( j \), we assume that (iii) the tenant will bid a price \( b^i_j \) for a fixed number \( s^i_j \) of spot instances. At the \( i \)-th allocation of \( j \), \( b^i_j \) together with the spot prices determines whether \( j \) can successfully obtain spot instances and how long it can utilize them. Since there is no exact statistical knowledge of spot prices over time under the online learning approach used in this paper and we only know that spot instances are on average cheaper than its on-demand counterpart, we assume that (iv) at every allocation the tenant will bid for the maximum number of spot instances under the parallelism constraint, i.e., \( s^i_j = \delta_j = o^i_j \). Here, we therefore have an important question of 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 \( z^i_j \) the remaining workload of \( j \) to be processed, i.e., \( z_j \) minus the workload of \( j \) that has been processed, where \( z^1_j = z_j \), and by \( \delta^i_j = \frac{(L - (i - 1) \cdot \text{Len}) \cdot \delta_j}{z^i_j} \) the current slackness of \( j \). Let \( s^i_j = \delta^i_j \).

The slackness of a job is used to measure the time flexibility that \( j \) has to allocate resources to \( j \) by the deadline and the process of allocating on-demand and spot instances to \( j \) is in fact divided into two phases with 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 \( s^i_j + 1 \geq 1 \);
- \( j \) does not have such flexibility at the next allocation update, otherwise.

An hour contains \( \text{Len} = \frac{60}{T} \) slots. As illustrated in Fig. 1, \( z^1_j = 132 \) and at the 1st allocation update, \( o^1_j = s^1_j = 2 \); then \( z^2_j = 132 - 2 \cdot 12 - 2 \cdot 8 = 92 \). At the 2nd update, \( o^2_j \) and \( s^2_j \) are still 2 and then \( z^3_j = 92 - 2 \cdot 12 - 2 \cdot 8 = 52 \). Further, \( s^3_j = \frac{\text{Len} \cdot \delta_j}{z^3_j} < 1 \) and there is no flexibility for \( j \) to utilize unstable spot instances at the 3rd 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 \( i_j = 2 \).

Secondly, when reserved instances are taken into account, we assume that (v) the allocation of long-term instances to a job can be updated at most once at every allocation update of that job. We denote by \( r^i_j \) the number of long-term instances assigned to \( j \) at the \( i \)-th allocation. In this paper, \( o^i_j \) and \( s^i_j \) 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 III-C, the acquired on-demand instances may not be fully utilized for an entire hour at the \( i \)-th allocation, we use \( o^i_j(t), s^i_j(t) \) and \( r^i_j(t) \) to denote the numbers of on-demand, spot and reserved instances that are actually utilized by \( j \) at every slot \( t \in [a_j, a_j + d_j - 1] \), where \( r^i_j(t) = r^i_j \) for all \( t \in [a_j + (i - 1) \cdot \text{Len}, a_j + i \cdot \text{Len} - 1] \). The parallelism constraint further translates to \( o^i_j + s^i_j + r^i_j = \delta_j \) and \( o^i_j(t) + s^i_j(t) + r^i_j(t) \leq \delta_j \) when self-owned instances are also considered.

D. Characterizing the Cost and Objective

In this subsection, we formally characterize the costs of various utilization states of long-term, spot and on-demand instances while processing a job. Here, a user is assumed to have \( R \) self-owned instances and reserved instances are not taken into account temporarily.

Let \( D^i_j \) denote the total number of allocation updates of \( j \) until its completion; as illustrated in Fig. 1 \( D_j = 3 \). Let \( X^i_{j,1}, X^i_{j,2}, X^i_{j,3} \in \{0, 1\} \) denote the random variables indicating whether or not the spot instances are obtained successfully at the \( i \)-th allocation, where \( X^i_{j,1} + X^i_{j,2} + X^i_{j,3} = 1 \). In the case where the \( i \)-th allocation is not the last one (i.e., \( i < D_j \)), if the spot instances run for an entire hour at this allocation, (i) the spot price is charged and (ii) set \( X^i_{j,1} = 1 \) and set \( X^i_{j,2} \) and \( X^i_{j,3} = 0 \). In the case where the \( i \)-th allocation is the last one, if the spot instances run until the completion of \( j \), (i) the spot price is charged and (ii) set \( X^i_{j,2} = 1 \) and set \( X^i_{j,1} \)

---

3When addressed in Section IV-B they are viewed as another type of self-owned instances whose total fee for a long term (e.g., 1 year) would be fixed and charged before their use. Now, we don’t need to consider them.
and \( X_{j,3} \) to 0; let \( T_{j,1}^i \) denote the number of time slots during which \( j \) is utilizing spot instances, where \( 0 \leq T_{j,1}^i \leq Len \). If the spot price is not charged and the instances are interrupted by the cloud or are not obtained even at the beginning of this allocation, then set \( X_{j,1}^i = 1 \) and \( X_{j,1}^i = X_{j,2}^i = 0 \); let \( T_{j,2}^i \) denote the number of slots during which \( j \) is utilizing these spot instances, where \( 0 \leq T_{j,2}^i < Len \). Then, we have for all \( j \in J \) that the cost of completing a job \( j \) is

\[
c_j = \sum_{i=1}^{D_j} s_i^j \cdot p_i^j \cdot X_{j,1}^i + X_{j,2}^i + o_i^j \cdot p, \tag{1}
\]

subject to that \( j \) is completed by the deadline, i.e., \( z_j \) equals

\[
\sum_{i=1}^{D_j} (X_{j,1}^i \cdot Len + X_{j,2}^i \cdot T_{j,1}^i + X_{j,3}^i \cdot T_{j,2}^i) \cdot s_i^j + a_j + d_j - 1 \quad (2)
+ \sum_{t=a_j}^{d_j-1} o_i(t) + r_j(t),
\]

where \( p_i^j \) is the average spot price to be charged at the \( i \)-th allocation of \( j \). Let \( \gamma \) denote the average utilization of the self-owned instances and we have

\[
\gamma \sum_{j \in J} \max \{a_j + d_j - 1\} = \sum_{j \in J} \sum_{i=1}^{D_j} (X_{j,1}^i \cdot Len + X_{j,2}^i \cdot T_{j,1}^i) r_j^i. \tag{3}
\]

The total cost of completing all jobs of a user is

\[
\sum_{j \in J} c_j. \tag{4}
\]

The main objective of this paper is to minimize Expression (4). We set \( p' \) to

\[
\frac{\sum_{j \in J} \sum_{i=1}^{D_j} s_i^j p_i^j (X_{j,1}^i + X_{j,2}^i)}{\sum_{j \in J} \sum_{i=1}^{D_j} s_i^j p_i^j (X_{j,1}^i \cdot Len + X_{j,2}^i \cdot T_{j,1}^i + X_{j,3}^i \cdot T_{j,2}^i)}. \tag{5}
\]

In the long run, \( p' \) represents the average cost of utilizing spot instances and we make the following assumption:

\[
0 = p_1 < p' < p, \tag{6}
\]

where \( p \) is the on-demand price and this is also one motivation of bringing spot instances into the cloud market.

### E. Scheduling Policies

In Section II-C, we give the rules of allocating resource to jobs that determines the form of scheduling policies. These policies specify how many self-owned, spot and on-demand instances are utilized at each allocation to a job \( j \). Now, we clarify what information are available for us to specify \( r_j^i, o_j^i, a_j, d_j \) and the amount of available long-term 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 long-term instances.

### Definition 2

At the \( i \)-th allocation of \( j \), the policy is a function \( F : \mathcal{Y}_j^i \times \mathcal{N} \to \mathcal{r}_j^i, f \), where \( f \in [0, 1] \); in other words, the job \( j \) will be allocated \( r_j^i \) long-term instances, acquire \( (1 - f) \cdot (\delta_j - r_j^i) \) on-demand instances, and bid some price for \( f \cdot (\delta_j - r_j^i) \) spot instances.

A main aim of this paper is to specify the function \( F : \mathcal{Y}_j^i \times \mathcal{N} \to \mathcal{r}_j^i, f \) so as to minimize Expression (4), where only \( \mathcal{Y}_j^i \) and \( \mathcal{N} \) are known. Finally, the main notation of this paper is also summarized in Table [I].

### III. Scheduling Policies in the Cloud

In this section, we studies the expected optimal or effective policies to determine the numbers of self-owned, spot and on-demand instances utilized by jobs at each resource allocation, specifying the function \( F : \mathcal{Y}_j^i \times \mathcal{N} \to \mathcal{r}_j^i, f \) in Definition 2. Here, reserved instances are not taken into account.

#### A. Optimal Structure

We first give (i) the optimal order to utilize the three types of instances, and (ii) the objective the scheduling policy for each type of instances should achieve. Our objective is to minimize the total cost, i.e., Expression (4).

Due to Inequality (6) and Equation (3), increasing the utilization \( \gamma \) of self-owned instances can reduce the consumption of spot and on-demand instances as indicated by Equation (2), therefore decreasing Expression (4). Further, due to \( p' < p \), after self-owned instances are used up, we observe Equations (4) and Expression (5) and have that consuming as many spot instances as possible can further minimize Equation (4). Built on this, we conclude that

**Principle 1.** The scheduler should make long-term 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 long-term 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.

In this following, the slackness of jobs is used as a measure of the flexibility or opportunity that jobs have to utilize unstable spot instances. As we will see, the allocation of self-owned and on-demand instances has the capacity of adjusting the slackness of jobs. There is a threshold for each type of instances and the policies we propose will utilize them in a way such that (i) if the slackness of a job is above the threshold, \( j \) will not be allocated self-owned or on-demand instances, and (ii) if its slackness is below the threshold, the smaller the slackness is, the more \( j \) is allocated. Against the diversity of jobs or the dynamics of spot prices, online learning approach can learn the best thresholds to increase the slackness of all jobs while maintaining a good utilization of self-owned instances, which well realizes Principle 1 and Principle 2.
B. Scheduling Policy for Self-owned Instances

Let $N(t)$ denote the number of self-owned instances that are currently idle at a slot $t$; let $m_t(t_2) = \min \left\{ N(t_1), \ldots, N(t_2) \right\}$, where $t_1 \leq t_2$. The policy that we propose is as follows: upon arrival of a job $j$, it will immediately be allocated

$$r_j = (1 - \min \{\beta \cdot s_j, 1\}) \cdot \min \left\{ \frac{\nu}{\delta_j}, m_t(a_j + d_j - 1) \right\}$$

self-owned instances at every slot in $[a_j, a_j + d_j - 1]$ where $\beta \in [0, 1]$ (as in [5], the issue of rounding the allocations of a job to integers is ignored for simplicity). Now, we explain the reason why this policy approximately realizes Principle [1]. After the allocation of self-owned instances to job $j$, the completion of job $j$ can be viewed as processing a new job $j'$ where $\delta_{j'} = \delta_j - d_j$, $a_{j'} = a_j$, $d_{j'} = d_j$, and $z_{j'} = z_j - r_j \cdot d_j$. Here, when only spot and on-demand instances are available to process the jobs, $j'$ has more flexibility (i.e., a larger slackness) than $j$ to utilize spot instances since we have $1 \leq \frac{\delta_j}{\delta_j - d_j} \leq \frac{d_j - r_j}{d_j - r_j - d_j}$, where $\nu = r_j \cdot d_j$. As a result, the policy realizes the second point of Principle [1] approximately. As illustrated in Fig. 2, the smaller $\beta$ is, the more self-owned instances is allocated to $j$; accordingly, choosing a smaller $\beta$ (e.g., $\beta = 0$) can lead to a high utilization of self-owned instances since each job is allocated a larger proportion of self-owned instances and more of them are utilized.

However, choosing a small $\beta$ (e.g., close to 0) will lead to that jobs with both larger and smaller slackness obtain almost the same large proportion of self-owned instances. As a consequence, given the limited number of self-owned instances, it does not have the capacity to make (i) the jobs with smaller slackness (especially these jobs with slackness equal to 1) allocated more self-owned instances, with their slackness enlarged to have more opportunity to utilize spot instances, while (ii) jobs with large enough slackness obtain less self-owned instances. In contrast to this, choosing an appropriately large $\beta$ can enable the policy to have such capacity. In Fig. 2 when $\beta = 0.3$, a job with a slackness 1 can obtain 14 self-owned instances while a job with a slackness 3 can only get 2 self-owned instances. This is useful when there are a significant proportion of jobs with very smaller slackness and a significant proportion of jobs with larger enough slackness.

Online learning used in the next section will learn the most cost-effective $\beta$ so as to balance a high utilization of self-owned instances and the opportunity that all jobs have to utilize spot instances. As we will see, the above policy for self-owned instances also enables analyzing the optimal principle to utilize spot and on-demand instances in a common case in Section III-C and bringing a stable cost structure to propose a solution for reserved instances in Section IV-B.

C. Cost-Effective Policy for Spot Instances

Once a job $j$ is allocated $r_j$ reserved 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. We first analyze the optimal policy to utilize on-demand in the case where there

| Symbol | Explanation |
|--------|-------------|
| $L$    | length of a time slot (e.g., 5 minutes) |
| $Len_j$| the number of time slots in an hour, i.e., $\frac{60}{L}$ |
| $J$    | a set of jobs that arrive over time |
| $j$ and $a_j$ | a slot of $J$ 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 |
| $\delta_j$ | the parallelism bound, i.e., the maximum number of instances that can be simultaneously used by $j$ |
| $s_j$  | the slackness, i.e., $\frac{\delta_j}{z_j}$, where $z_j/\delta_j$ denotes the minimum execution time of $j$ |
| $T$    | the number of time slots, i.e., $\max_{j \in J} \{a_j\}$ |
| $D_j$  | the total number of allocations to $j$ until its completion |
| $s_{i,j}$ | the number of spot instances bid for at the $i$-th allocation update of $j$ |
| $o_{i,j}$ | the number of on-demand instances acquired for $j$ at its $i$-th allocation update |
| $r_{j,i}^s$, $s_{i,j}^s$, and $o_{i,j}^s$ (resp. $r_{j,i}^p$, $s_{i,j}^p$, and $o_{i,j}^p$) | the number of self-owned, spot and on-demand instances utilized by $j$ at a slot $t$ |
| $p_j^s$ | the bid price at the $i$-th allocation of $j$ |
| $p_j^p$ | the spot price charged at the $i$-th allocation of $j$ |
| $z_j^s$ | the remaining workload of $j$ to be processed at the $i$-th allocation update to $j$ |
| $s_j^t$ | the slackness at the $i$-th allocation update, i.e., $(d_j - (s_j - t \cdot L_{en}) - \delta_j)$ |
| $\gamma$ | the resource utilization of long-term instances |
| $X_{j,1}^s, X_{j,2}^s, Y_{j,1}^s, Y_{j,2}^s$ | random variables characterizing the state in which spot instances are utilized by $j$ at the $i$-th allocation (see Paragraph 2 of Section II-D for more explanation.) |
| $J_0'$ | a set of policies, each indexed by $\pi$ |
| $\{\beta, \beta_1, \beta_2, b\}$ | a tuple of parameters that defines a policy and determines the allocation of various instances to $j$ at each allocation |
| $f(s_j^t)$ | a function defined by $\min \{\{\beta_1 \cdot s_j^t\}^{\beta_2}, 1\}$ and determining the proportion of spot instances |
| $T_0$ | a period in which a set of jobs $J_0'$ recur |
| $r_j$ | the number of long-term instances allocated to a job $j$ at every $t \in [a_j, a_j + d_j - 1]$ |
| $J_0'$ | after the allocation of long-term instances to $J_0'$, the virtual jobs to be processed by spot and on-demand instances alone, each with an arrival time $a_j$, a deadline $d_j$, a workload $z_j - r_j \cdot d_j$, and a parallelism bound $\delta_j - r_j$ |
is no flexibility to utilize unstable instances that is defined in Definition \(1\).

Let \( \kappa = \left[ \frac{a_j + d_j - 1}{\text{Len}} \right] \), denoting the maximum integer multiple of an hour (containing \( \text{Len} \) slots) in \([t''+1, a_j + d_j - 1]\), and \( t'' = a_j + d_j - \kappa \cdot \text{Len} \) where \( 0 < t'' - 1 < t'' < \text{Len} \). Given the fact that on-demand instances are charged on an hourly basis, we conclude that

**Proposition 1.** Once spot instances are terminated by the cloud at the \( i \)-th allocation, in the case where \( j \) has no flexibility to utilize spot instances at the next allocation update, the cost-optimal strategy to utilize on-demand instances is to

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

**Proof.** We prove this by contradiction using the fact that on-demand instances are charged on an hourly basis. There are \( \delta_j \) instances and consider the workload of \( j \) to be processed on each instance over the slot interval \([t''+1, a_j + d_j - 1]\). Given any allocation \( A \) of \( j \) over \([t''+1, a_j + d_j - 1]\). Let \( \chi_k \) denote the total amount of workload processed at the \( k \)-th instance (\( 1 \leq k \leq \delta_j \)). The allocation \( A \) as illustrated by Fig. 3 (left) can be transformed into an allocation \( A' \) with the following form without increasing the total cost of utilizing on-demand instances: the \( \chi_k \) workload of the \( k \)-th instance is processed from the deadline \( a_j + d_j - 1 \) towards earlier slots until the slot \( a_j + d_j - \kappa \). The transformed allocation is also illustrated in Fig. 3 (middle).

Now, we only need to show the cost-optimal structure when the allocation is of the transformed form illustrated in Fig. 3 (middle). Utilizing an on-demand instance for a partial hour is not cost-optimal if there is the opportunity to utilize it for the entire hour. In the transformed allocation \( A' \), if there is an instance that is not fully utilized in the interval \([t'', a_j + d_j - 1]\), the cost-optimal strategy is to make that interval fully utilized and, correspondingly, the workload of \( j \) that is processed on \( \delta_j \) machines in \([t''+1, t''-1]\) is reduced equivalently. Let \( \tau_j = z_j^{i+1} - \text{Len} \cdot \delta_j \cdot \kappa \) denoting the remaining workload to be processed in \([t''+1, t''-1]\) under the above allocation in \([t'', a_j + d_j - 1]\), and let \( \kappa' = \frac{\tau_j}{\text{Len}} \). In the interval \([t''+1, t''-1]\), the optimal strategy is acquiring \( \kappa' \) on-demand instances to process the remaining workload where each instance is utilized for a partial hour. The final optimal allocation is also illustrated in Fig. 3 (right).

In the following, we analyze the expected cost-optimal or effective policy in two cases when there is the flexibility for \( j \) to utilize unstable spot instances.

**Independent Case.** We first study the (independent) case where the change of spot prices is independent of the jobs’ arrival of a user; so the distributions over \( X_{j,1}^i, X_{j,3}^i, T_{j,2}^i \) are irrelevant of \( j \) and \( i \) for all \( j \in \mathcal{J} \) and \( i \in [1, D_j - 1] \).

Let \( P(x_{j,1}^i, x_{j,2}^i, x_{j,3}^i, t_{j,1}^i, t_{j,2}^i) \) denote the probability when \( X_{j,1}^i = x_{j,1}^i, X_{j,2}^i = x_{j,2}^i, X_{j,3}^i = x_{j,3}^i, T_{j,1}^i = t_{j,1}^i, \) and \( T_{j,2}^i = t_{j,2}^i \) at the \( i \)-th allocation of \( j \). In the allocations to \( j \) excluding the last one, we have \( \sum_{i=0}^{\text{Len}-1} P(0, 0, 1, 0, t) + P(1, 0, 0, 0, 0) = 1 \); in the last allocation, we assume that the instances will have run for \( \tau \) time slots at the completion time of \( j \), where \( 0 < \tau \leq \text{Len} \), and we have \( \sum_{i=0}^{\tau-1} P(0,0,1,0, t) + P(0,1,0, \tau,0) = 1 \). The expected workload processed by spot instances at \( i \)-th allocation is \( t_j^i \cdot s_j^i \), where the expected processing time by spot instances \( t_j^i = \sum_{i=0}^{\text{Len}-1} t \cdot P(0,0,1,0, t) + \text{Len} \cdot P(1,0,0,0,0) \) at the allocations excluding the last one (i.e., \( i \in [1, D_j - 1] \)) and \( t_j^i = \sum_{i=0}^{\tau-1} t \cdot P(0,0,1,0, t) + \tau \cdot P(0,1,0, \tau,0) \) at the last allocation (i.e., \( i = D_j \)). The bid price together with spot prices determines the probability \( P(x_{j,1}^i, 0, x_{j,3}^i, 0, t_{j,2}^i, \epsilon) \), i.e., whether a user can successfully obtain spot instances (s)he bids for and how long \( j \) could utilize them. In the independent case, the expectation \( t_j^i \) of the time of utilizing spot instances at each allocation is assumed to be \( \omega \) for all \( i \in [1, D_j - 1] \) and the following proposition shows that Principle 2 can be optimally realized.

**Proposition 2.** In the independent case, the expected optimal policy to utilize spot instances is that, \( \delta_j \) spot instances are bid for at every allocation until there is no flexibility for \( j \) to utilize safe on-demand instances.

**Proof.** If there is the flexibility for \( j \) to utilize spot instances as defined in Definition 1 at every allocation update of \( j \), Proposition 2 holds trivially. In the following, we consider the case where there exists an allocation update from which on there is no flexibility to utilize spot instances. Given an allocation of \( j \) over \([a_j, a_j + d_j - 1]\), the expected periods of utilizing spot instances at different allocation updates \( i \) are the same where \( 1 \leq i \leq i_j \) in the independent case. At the first \( i_j \) allocations, the expected amount of workload processed by spot instances is only determined by \( \sum_{k=1}^{i_j} s_{j,k}^i \). Let \( \varepsilon = [\frac{\sum_{k=1}^{i_j} s_{j,k}^i}{\sum_{k=1}^{i_j} s_{j,k}^i}] \). Hence, any allocation of \( j \) over \([a_j, a_j + d_j - 1]\) (illustrated by Fig. 3 (left)) can be finally transformed into an allocation of the following form such that the expected amount of workload processed by spot instances does not change (illustrated in Fig. 4 (middle)): (i) \( \delta_j \) spot instances are bid for at every allocation update until the \( \varepsilon \)-th allocation update, and (ii) \( \sum_{k=1}^{i_j} s_{j,k}^i - \varepsilon \cdot \delta_j \) spot instances are bid for at the \((\varepsilon + 1)\)-th allocation update. For the transformed allocation over \([a_j, a_j + d_j - 1]\) illustrated by Fig. 4 (middle), the optimal strategy to utilize spot instances is to bid for \( \delta_j \) spot instances at the \((\varepsilon + 1)\)-th allocation update as illustrated in Fig. 4. Hence, Proposition 2 holds.

In the following, we discuss the correlated case.

**Correlated Case.** Due to the diversity of users, sometimes, the assumption in Proposition 2 may be violated and the jobs’ arrival of a user correlated to the change of spot prices. This will lead to that the distributions over \( X_{j,1}^i, X_{j,3}^i, T_{j,2}^i \) are relevant to \( j \) or \( i \) where \( 1 \leq i \leq D_j - 1 \).

Now, we analyze what event may harm the cost efficiency in this case. If a job can utilize lots of spot instances at the
is proposed to determine how many on-demand and spot instances at the 1st, 2nd and 3rd allocation updates.

Fig. 3. Illustration for the Proof of Proposition 1 where \( \delta_2 = 4 \), the green (resp. blue) areas denote the workload of \( j \) processed by spot (on-demand) instances at the 1st, 2nd and 3rd allocation updates.

Fig. 4. Illustration for the Proof of Proposition 2 where \( \delta_2 = 4 \), the green (resp. blue) areas denote the workload of \( j \) processed by spot (on-demand) instances at the 1st and 2nd allocation updates.

earliest allocations of \( j \), i.e., \( X^{j,1}_i = 1 \) and \( t^{j,0}_i = \text{Len} \) at the earliest allocation of \( j \), then this will not harm the cost efficiency. In the opposite, for substantial jobs \( j \) of a tenant, it happens that \( X^{j,1}_i = 0 \) and \( t^{j,0}_i \) in the earliest allocations of \( j \) are much smaller than the later ones. In this case, if \( s^{j}_i \) is set to a larger value (e.g., \( \delta_2 \)), the job will quickly get into a situation in which it has to totally utilize more expensive on-demand instances to ensure that \( j \) is completed by the deadline, and has no opportunity to utilize the cheaper spot instances in the later allocations. This harms the cost efficiency of utilizing spot and on-demand instances.

Let \( \beta_1 = \sum_{j \in \{\text{on-demand}\}} \frac{\beta_j}{\text{Len} \cdot (\text{Len} - 1) + \gamma} \), i.e., the ratio of the period in which spot instances are utilized to the whole period in which \( j \) is processed. Accordingly, we propose the following policy to determine the proportion of spot instances at every allocation:

\[
\text{f}(s^{j}_i) = \min \left\{ (\beta_1 \cdot s^{j}_i)^{\beta_2}, 1 \right\},
\]

where \( \beta_1 \in (0, 1) \) and \( \beta_2 \in [0, +\infty) \). Now, we explain why using Equation (7) can near-optimally realize Principle 2 in the correlated case. If \( s^{j}_i \geq \frac{\delta_2}{\beta_2} \), \( j \) will bid for \( \delta_j \) spot instances; if \( s^{j}_i < \frac{\delta_2}{\beta_2} \) and \( \beta_2 \) is large enough, \( \text{f}(s^{j}_i) \) is close to 0. Hence, for jobs with relatively small slackness, selecting appropriate \( \beta_1 \) and large enough \( \beta_2 \) can guarantee that, \( j \) will totally utilize (or utilize a large proportion of) on-demand instances at the early allocations, and, as its current slackness becomes larger, it will utilize a larger proportion of spot instances in the later allocations. Here, we still have that \( 1 \leq \frac{\delta_j}{\beta_2} \cdot (d_j - (i - 1) \cdot \text{Len}) \leq \frac{\mu \cdot s^{j}_i}{\beta_2} \cdot \text{Len} \), where \( \mu = \delta_j \cdot (d_j - (i - 1) \cdot \text{Len}) \) and \( \nu = s^{j}_i \cdot \text{Len} \). When online learning is applied, the best parameters \( \beta_1 \) and \( \beta_2 \) are learned so that when the slackness of jobs is smaller than some threshold, the smaller the slackness is, the more on-demand instances \( j \) will utilize. This guarantees that, at the later allocation updates of \( j \), it has a better opportunity to utilize spot instances.

Finally, with Proposition 2 and Equation (7), Algorithm 1 is proposed to 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.

Algorithm 1: Proportion \((j, \beta_1, \beta_2, b)\)

### Input
- at a slot \( t \), a job \( j \) to be processed with the characteristics \( \{a_j, d_j, z_j^L, \delta_j\} \) and a parameterized policy \( \{\beta, \beta_1, \beta_2, b\} \)

### Algorithm
1. if the independent case happens then
   2. \( f' \leftarrow 1 \);
2. else if the correlated case happens then
   3. \( f' \leftarrow \text{f}(s^{j}_i) \);
4. for \( t \leftarrow t + \frac{\text{Len}}{b} - 1 \) do
   5. \( \text{si}_j(t) \leftarrow f' \cdot \min \left\{ \frac{\delta_j - r_j(t)}{\min(d_j - L - (i-1) \cdot \text{Len})}, \frac{\delta_j - r_j(t)}{\min(d_j - L - (i-1) \cdot \text{Len})} \right\} \); 
   6. \( \text{ao}_j(t) \leftarrow \delta_j - \text{si}_j(t) - r_j(t) \);
   7. \( \text{bi}_j \leftarrow 0 \);
9. at the \( i \)-th allocation update, bid a price \( \text{bi}_j \) for \( \text{si}_j(t) \) spot instances;

Algorithm 2: Dynalloc

### Input
- at a slot \( t \), a job \( j \) to be processed with the characteristics \( \{a_j, d_j, z_j^L, \delta_j\} \) and a parameterized policy \( \{\beta, \beta_1, \beta_2, b\} \)

### Algorithm
1. if \( a_j = t \) then
   2. // upon arrival of \( j \), allocate long-term instances to it
   3. \( r_j \leftarrow (1 - \min \{\beta \cdot s^{j}_i, 1\}) \cdot \min\left\{\frac{\delta_j}{\beta_2} \cdot (d_j - (i - 1) \cdot \text{Len}), \mu \right\} \);
   4. for \( t \leftarrow a_j \) to \( a_j + d_j - 1 \) do
   5. \( r_j(t) \leftarrow r_j \);
   6. \( i \leftarrow t - a_j + 1 \);
   7. if \( t - a_j = i - 1 \) then
   8. // at the \( i \)-th allocation update of \( j \), \( j \) has the flexibility for spot instances
   9. call Algorithm 1;
   10. if the spot instances of \( j \) are terminated at \( t - 1 \) then
   11. // \( j \) has no flexibility to utilize spot instances at the next allocation update
   12. apply the strategy in Proposition 2 here;
   13. otherwise, \( j \) still has the flexibility at the next allocation update where \( z_j^{L+1} = z_j^L + 1 \);

D. Scheduling Framework

As described above, a general policy is defined by a tuple \( \{\beta, \beta_1, \beta_2, b\} \) and determines the amount of long-term instances and the proportions of spot and on-demand instances allocated to a job, and the bid price for spot instances. This also specifies the function \( F \) in Definition 2. 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 \( t \). The complete framework used to determine the allocation of self-owned, spot
and on-demand instances to every such job j at t is presented as Algorithm 2 where $z_j$ always denotes the remaining workload of j to be processed after deducting its current allocation from the total workload of j. This algorithm will check the state of j to decide how to allocate computing instances to it.

IV. APPLICATION OF ONLINE LEARNING

In this section, online learning is applied to purchase reserved, spot, and on-demand instances.

A. Self-owned, Spot, and On-demand Instances

In this subsection, spot and on-demand instances are considered for purchasing with self-owned instances available. The online learning algorithm that we adopt is the one in [5], 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 $\{\beta, \beta_1, \beta_2, b\}$ in Section III-D. Let $d = \max_{t \in \mathcal{T}} |\{d_j\}|$ and $\mathcal{J}_t \subseteq \mathcal{J}$ denote the jobs j with $a_j = t$. When a job j in $\mathcal{J}_t$ arrives, the algorithm randomly picks a policy according to a distribution and bases the resource allocation on the job that on 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, this enables computing the expected cost of each policy (i.e., its probability) so that the weight of each policy (i.e., its probability) is updated so that the lower-cost (higher-cost) policies are re-assigned the enlarged (resp. reduced) weights.

Algorithm 3: OptiLearning

| Input | a set $\mathcal{P}$ of n policies, each $\pi$ being parameterized so that $\pi \in \{1, 2, \ldots, n\}$; the set of jobs $\mathcal{J}_t$ that arrive at t; |
|-------|---------------------------------------------------------------|
|       | $w_j = \{w_{j,1}, \ldots, w_{j,n}\} = \{1/n, \ldots, 1/n\};$ |
| for t $\leftarrow$ 1 to $T$ do |
| if $\mathcal{J}_t \neq \emptyset$ then |
| for each $j \in \mathcal{J}_t$, pick a policy $\pi$ 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 |
| $\eta_t \leftarrow \frac{\sum_{j \in \mathcal{J}_{t-d}} \gamma_j}{|\{j \in \mathcal{J}_{t-d}\}|}$; |
| get a job $j$ from $\mathcal{J}_{t-d}$; |
| for $\pi \leftarrow 1$ to n do |
| $w_{j+1, \pi} \leftarrow w_{j, \pi} e^{\eta_t \gamma_j (\pi)}$; |
| for $\pi \leftarrow 1$ to n do |
| $w_{j+1, \pi} \leftarrow \frac{w_{j+1, \pi}}{\sum_{i=1}^{n} w_{j+1, i}}$; |
| $\mathcal{J}_{t-d} \leftarrow \mathcal{J}_{t-d} - \{j\}$; |
| end |
| end |

As modeled in Section 2, when an online learning algorithm runs, the cost of completing a job is from the use of spot and on-demand instances alone; so, the cost $c_j(\pi_j)$ of completing a job j under some policy $\pi_j \in \mathcal{P}$ is defined by Equation (1), i.e., the cost incurred by the consumed spot and on-demand instances excluding self-owned instances. Only if we apply the scheduling framework (i.e., Algorithm 2) to the online learning approach, the policy in $\mathcal{P}$ with the lowest cost is also the policy with the best configuration parameters $\{\beta, \beta_1, \beta_2, b\}$ to realize Principles [1] and [2] and minimize Equation (4). As proved in [5], we have that

Proposition 3. Let $\mathcal{J}_t$ denotes the jobs that arrive at time slot t, $N' = \bigcup_{t=d+1}^{T} \mathcal{J}_t$, and $d = \max_{t \in \mathcal{T}} \{d_j\}$. For all $\delta \in (0, 1)$, it holds with a probability at least $1 - \delta$ over the random of online learning that

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

B. Case Incorporating Reserved Instances

Reserved instances have a long-term commitment for at least 1 year and they can be considered for purchasing on the condition that a set of jobs $\mathcal{J}_0$ recurs in a period of length $T_0$, which is the case of many scenarios [16], [17]. This condition enables a stable cost structure to predict the expected cost of utilizing spot and on-demand instances. Denote by $R$ the total number of two types of long-term instances and these long-term instances are allocated using Algorithm 2. The number of long-term instances allocated to every job $j \in \mathcal{J}_t$ at every $t \in [a_j, a_j + d_j - 1]$ is a constant $r_j$, which depends only on $R$ and $\mathcal{J}_0$, and, the completion of j can be viewed as processing a new job $j'$ with $\delta_j' = \delta_j - r_j$, $a_j' = a_j$, $d_j' = d_j$, and $z_j' = z_j - r_j \cdot d_j$, where only spot and on-demand instances are available for processing. Here, long-term instances have a fixed utilization of $\gamma = \frac{\sum_{j \in \mathcal{J}_0} r_j \cdot d_j}{(R \cdot T_0)}$ every period. Let $\mathcal{J}_t'$ denote the set of all the new virtual jobs $j'$, where $\mathcal{J}_0'$ depends on $\mathcal{J}_0$ and R. Assume that $R$ is an upper bound of reserved instances that need to be purchased, e.g., $R = \max_{1 \leq t \leq T} \{\sum_{j \in \mathcal{J}(t)} \delta_j\}$, where $\mathcal{J}(t) = \{j \in \mathcal{J}|t \in [a_j, a_j + d_j - 1]\}$.

Since the statistical characteristics of spot prices do not change much every year [13], [18] and the performance of online learning is close to that of the best policy, the final solution is presented as Algorithm 4 to approximately determine the optimal amount of reserved instances $R_2^*$ to be purchased. Here, $\mathcal{J}(t)$ is used as the expected cost of utilizing spot and on-demand instances alone to complete $\mathcal{J}_0'$; the allocation of long-term instances is irrelevant to spot prices and, given R, has a stable effect on the feature of the jobs (i.e., $\mathcal{J}_0'$) to be processed by spot and on-demand instances. Once $R_2^*$ is determined by Algorithm 4, we can run Algorithm 3 to process the jobs that arrive in the time to come; using Algorithm 3 to choose policies for jobs is also more robust against the future cloud market dynamics than using the best policy alone for processing all jobs and Algorithm 3 will again approach the performance of the best policy.

V. EVALUATION

The behaviors of Algorithm 3 over real datasets basically coincide with its behaviors over synthetic datasets [5] and

4Here, $T_0$ needs to be so large that the dynamics of spot prices can be reflected sufficiently in a time interval of length $T_0$. 
Algorithm 4: ResAmount

**Input**: the information on spot prices, on-demand price, \( J'_0 \), the best policy \( \pi^* \) from \( \mathcal{P} \)

1. without loss of generality, the starting time slot is set to 1;
2. use Algorithm 2 to compute the cost \( c_{j'}(\pi^*) \) of completing \( j' \in J'_0 \) under the policy \( \pi^* \);
3. \( \tau(J'_0) \leftarrow \sum_{j' \in J'_0} c_{j'}(\pi^*) \);
4. \( R_2^* \leftarrow \min_{1 \leq R_2 \leq \tau} (p_2 \cdot R_2 + \sum_{j' \in J'_0} c_{j'}(\pi^*)) \);

synthetic datasets can provide the evaluation with diverse workloads and spot prices so as to validate the effectiveness or correctness of the proposed policies, solutions and conclusions; so we only take the simulations on synthetic datasets.

### A. Parameter Settings

The on-demand price is \( p = 0.25 \) per hour while reserved instances bring a discount of 50% compared to on-demand instances [3]. All the jobs in the experiments have a parallelism bound of 20. The first type of jobs and spot prices is generated as follows: (i) the job’s arrival is generated according to a Poisson distribution with a mean of 10 minutes. The size of each job is chosen uniformly from the interval \([0, 600]\). The job’s deadline is generated as \( x \cdot z_j/\beta_j \), where \( x \) is uniformly distributed over \([1, 3]\), (ii) spot prices are updated every time slot \([13]\). Their values are chosen independently as \( 0.15 + \max\{x_2, 0\} \), where \( x_2 \) is a normal random variable with a mean \( \mu = 0 \) and a variance \( \sigma = 1 \). Here, the change of spot prices is independent of the arrival of jobs over time, which is the case in Proposition [2].

The second type of jobs and spot prices is generated as follows: (i) jobs are generated periodically: the size \( z_{2i+1} \), arrival time \( a_{2i+1} \) and deadline \( d_{2i+1} \) of \((2i + 1)\)-th job are \( 780, 96i + 1 \) and 48; the \( z_{2i+2}, a_{2i+2} \) and \( d_{2i+2} \) of \((2i + 2)\)-th job are set to \( 480, 96i + 49 \) and 48 for all \( i = 0, 1, 2, \ldots \); (ii) for each of the first 12 time slots respectively in \([d_{2i+1}, a_{2i+1} + d_{2i+1} - 1] \) and in \([a_{2i+2}, a_{2i+2} + d_{2i+2} - 1] \), the spot price is set to \( 0.21 + 0.05 \cdot \max\{0, x_3\} \), where \( x_3 \) is a normal random variable with \( \mu = 0 \) and \( \sigma = 3 \); For each of the rest time slots, it is set to \( 0.05 + 0.05 \cdot \max\{0, x_4\} \) where \( x_4 \) is a normal random variable with \( \mu = 0 \) and \( \sigma = 1 \). Here, the change of spot prices is correlated to the arrival of jobs over time, which is an example of the case for the robust policies in Section [II-C].

The parameters \( \beta, \beta_1, \beta_2, \) and \( b \) define the scheduling policies. \( \beta, \beta_1, \beta_2 \) are chosen respectively from \( \{0.0, 0.25, 0.35, 0.4\} \), \( \{0.4, 0.8\} \) and \( \{0.1, 20, 400\} \); the bid price \( b \in \{0.16, 0.19, 0.22, 0.25\} \). Here, \( \{\beta, \beta_1 = 0.4 \text{ or } 0.8, \beta_2 = 0, b\} \) define the same policy where \( f(s_j^i) = 1 \). Moreover, in [3], the algorithm will randomly select a parameter \( \theta \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\} \) and acquire a fixed proportion \( \theta \) of spot and on-demand instances from the cloud for every processed job \( j \).

### B. Results

**Experiment 1**. This experiment aims to validate the correctness of Proposition [2] and is taken under the first type of input where spot and on-demand instances are considered alone.

We refer to the ratio of the total cost of utilizing various instances to the total workload processed by them as the average unit cost. The minimum unit cost of the proposed policies, specified by \( \{\beta_1, \beta_2, b\} \), is 0.1921; the best policy is defined by \( \{\beta_1 = 0.4 \text{ or } 0.8, \beta_2 = 0, b = 0.22\} \) where \( f(s_j^i) = 1 \). This validates Proposition [2].

![Fig. 5. The average unit cost of each policy under the first type of input.](image)

The average unit cost of each policy is illustrated in Figure 6. The minimum unit cost of the proposed policies, specified by \( \{\beta_1, \beta_2, b\} \), is 0.1144; the optimal policy is the 27-th policy where \( \{\beta_1, \beta_2, b\} = \{0.8, 400, 0.25\} \) and \( f(s_j^i) = (0.8 \cdot s_j^i) \). Using the policies in [3], the best policy is the 24-th policy where \( \{\theta, b\} = \{1, 0.25\} \) with an average unit cost of 0.1762. The performance of the best policy of this paper improves the one in [3] by 35.07%. Starting with an initial distribution \( \{1/n, \ldots, 1/n\} \) over them, Algorithm 3 achieves an average unit cost of 0.1218 using the \( n = 32 \) policies of this paper while it achieves an average unit cost of 0.1813 using the \( n = 24 \) policies in [3]. The proposed policies of this paper improve the performance of Algorithm 2 by 32.82%, which shows the effectiveness of the proposed scheduling policies for spot and on-demand instances. With more and more jobs (about 4000 jobs) processed, the policies with the best performance (i.e., the 6th, 13th, 20th, and 27th policies in Figure 6) are assigned the highest probabilities; so Algorithm 3 has a performance close to the best policy in hindsight.

**Experiment 2**. This experiment aims to validate the effectiveness of the proposed robust policies for spot and on-demand instances in Section III-C and is taken under the second type of input where \( R = 0 \).

![Fig. 6. The average unit cost of each policy under the second type of input: the blue stars and green circles respectively correspond to the performance of the policies proposed in this paper and [3].](image)
After the allocation of long-term instances to a set of jobs \( J \) under a policy defined by \( \beta \), the jobs to be processed by spot and on-demand instances alone can be viewed as a new set of jobs \( J' \). The experimental results over 60000 time slots are shown in Table II. Here, \( \gamma \), \( \alpha_0 \), and \( \alpha \) denote the utilization of long-term instances, the minimum average unit cost of policies (defined by different \( \{\beta, \beta_1, \beta_2, b\} \)) to process \( J' \), the minimum average unit cost of policies to process \( J \) respectively when \( \beta = 0, 0.25, 0.35 \), and 0.4. In the experiment, the larger \( \beta \) is, the smaller \( \alpha_0 \) is. The limited long-term instances, choosing a larger \( \beta \) can reduce the allocation of them to jobs with larger slackness so as to allocate more of them to jobs with smaller slackness, which leads to that most of the jobs in \( J' \) have relatively larger slackness and more opportunity to utilize cheaper spot instances. However, a larger \( \beta \) can also lower the utilization of long-term instances greatly when the jobs arrive sparsely (every 10 minutes), thus leading to a larger \( \alpha \). The best policy is defined by \( \beta = 0, f(s_i^j) = 1 \), and \( b = 0.22 \), where \( b \) determines the probability that a job obtains spot instances. We still run Algorithm 3 over 60000 time slots to see how much is saved when purchasing 122 reserved instances; then, the average unit cost is 0.1303. This cost is close to the cost of the best policy and saves the average unit cost by 14.89%, compared with the average unit cost \( \alpha \) of Experiment 3, given in Table III, which shows that purchasing reserved instances can further reduce the cost of completing all jobs of a user; in Experiment 3, reserved instances are not considered.

### VI. Conclusion

The problem of cost-effective resource provisioning in IaaS clouds is a premier concern for all users. In this paper, we aim to provide a complete understanding of the problem of applying online learning to this problem. In particular, we extend a previous online learning approach, where on-demand and spot instances are considered, to the case also incorporating reserved instances and the instances possessed by a user itself. After modeling the effect of the pricing models on the instance utilization state, we also characterize the expected cost-optimal scheduling principles to utilize spot and on-demand instances and the instances possessed by a user itself, which is of great importance to the performance of the online learning approach. Finally, simulations are done to validate the correctness or effectiveness of the proposed conclusions and solutions in this paper. In the future, we are to add more simulations over datasets with more extensive parameter settings.

### References

[1] https://www.srgresearch.com/articles/cisco-maintains-lead-public-cloud-infrastructure-while-hp-leads-private.

[2] [http://softwarestrategies1blog.com/2011/06/01/sizing-the-public-cloud/](http://softwarestrategies1blog.com/2011/06/01/sizing-the-public-cloud/)

[3] [https://aws.amazon.com/ec2/purchasing-options/](https://aws.amazon.com/ec2/purchasing-options/)

[4] Oma Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, Dan Tsafrir. Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation*, 2013.

[5] 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.
[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] Yu-Ju Hong, Jiachen Xue, Mithuna Thottethodi. Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud. In ACM SIGMETRICS, 2011.
[8] Sivadon Chaisiri, Bu-Sung Lee, Dusit Niyato. Optimization of Resource Provisioning Cost in Cloud Computing. IEEE Transactions on Services Computing, 2012.
[9] Ruben Van den Bossche, Kurt Vanmechelen, Jan Broeckhove. IaaS Reserved Contract Procurement Optimisation with Load Prediction. Future Generation Computer Systems, 2015.
[10] Wei Wang, Baochun Li, Ben Liang. Optimal Online Multi-Instance Acquisition in IaaS Clouds. IEEE Transactions on Parallel and Distributed Systems, 2015.
[11] Shengkai Shi, Chuan Wu, Zongpeng Li. "Cost-Minimizing Online VM Purchasing for Application Service Providers with Arbitrary Demands." In IEEE CLOUD, 2015.
[12] Longbo Huang, Xin Liu, Xiaohong Hao. "The power of online learning in stochastic network optimization." In ACM SIGMETRICS, 2014.
[13] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, Xinyu Wang. How to Bid the Cloud. In ACM SIGCOMM, 2015.
[14] Yang Song, Murtaza Zafer, Kang-Won Lee. "Optimal Bidding in Spot Instance Market." In IEEE INFOCOM 2012.
[15] Murtaza Zafer, Yang Song, Kang-Won Lee. "Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs." In IEEE CLOUD, 2012.
[16] Arijit Khan, Xifeng Yan, Shu Tao, Nikos Anerousis. Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach. In IEEE NOMS, 2012.
[17] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayananurthy et al. "Morpheus: Towards Automated SLOs for Enterprise Clusters." In USENIX OSDI, 2016.
[18] Prateek Sharma, David Irwin, Prashant Shenoy. How Not to Bid the Cloud. In USENIX HotCloud, 2016.