A Framework for Allocating Server Time to Spot and On-demand Services in Cloud Computing

XIAOHU WU*, Nanyang Technological University, Singapore
FRANCESCO DE PELLEGRINI, University of Avignon, France
GUANYU GAO, Nanyang Technological University, Singapore
GIULIANO CASALE, Imperial College London, United Kingdom

Cloud computing delivers value to users by facilitating their access to servers in periods when their need arises. An approach is to provide both on-demand and spot services on shared servers. The former allows users to access servers on demand at a fixed price and users occupy different periods of servers. The latter allows users to bid for the remaining unoccupied periods via dynamic pricing; however, without appropriate design, such periods may be arbitrarily small since on-demand users arrive randomly. This is also the current service model adopted by Amazon Elastic Cloud Compute. In this paper, we provide the first integral framework for sharing the time of servers between on-demand and spot services while optimally pricing spot service. It guarantees that on-demand users can get served quickly while spot users can stably utilize servers for a properly long period once accepted, which is a key feature to make both on-demand and spot services accessible. Simulation results show that, by complementing the on-demand market with a spot market, a cloud provider can improve revenue by up to 461.5%. The framework is designed under assumptions which are met in real environments. It is a new tool that other cloud operators can use to quantify the advantage of a hybrid spot and on-demand service, eventually making the case for operating such service model in their own infrastructures.

CCS Concepts: • **Networks** → **Cloud computing**; **network economics**; • **Computing methodologies** → **Model development and analysis**; • **Mathematics of computing** → Queueing theory.

Additional Key Words and Phrases: cloud computing, spot and on-demand services, time allocation, pricing

ACM Reference Format:
Xiaohu Wu, Francesco De Pellegrini, Guanyu Gao, and Giuliano Casale. 2019. A Framework for Allocating Server Time to Spot and On-demand Services in Cloud Computing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 1, 1 (October 2019), 31 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The global cloud Infrastructure-as-a-Service (IaaS) market grew to $34.6 billion in 2017, and is projected to increase to $71.6 billion in 2020 [13]. IaaS delivers value to users by facilitating their access to servers or virtual machines: users can rent servers from cloud service providers (CSPs) whenever their need arises. From a CSP's perspective, an important question is what forms of services should be offered to attract more users and achieve high resource efficiency. One example

*Part of his work was done while he was with Fondazione Bruno Kessler, Trento, Italy.

Authors' addresses: Xiaohu Wu, Nanyang Technological University, Singapore, xiaohu.wu@ntu.edu.sg; Francesco De Pellegrini, University of Avignon, Avignon, France, francesco.de-pellegrini@univ-avignon.fr; Guanyu Gao, Nanyang Technological University, Singapore, ggao001@ntu.edu.sg; Giuliano Casale, Imperial College London, London, United Kingdom, g.casale@imperial.ac.uk.

2376-3639/2019/10-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

is Amazon Elastic Cloud Compute (EC2), the leading CSP, accounting for 51.8% of the global market share in 2017 [13]. There are two types of service: on-demand and spot instances (i.e., virtual machines) [2, 3]. The former are offered at a fixed price and users pay only for the period in which instances are consumed. They may be idle at times, and such states are sold in the form of spot instances to improve resource efficiency. Users can bid prices for spot instances that will be offered and run as long as their bid is above the spot price; spot users are billed by the spot price that is usually lower than the price of on-demand instances (i.e., on-demand price). So far, numerous works have been done for users to utilize these instances cost-efficiently [19, 20].

Offering users on-demand and spot services is an interesting option for CSPs. To understand the internal process, two intertwined aspects need to be controlled at once: (i) the sharing of server time among on-demand and spot users and (ii) the pricing of spot instances. A framework covering these aspects is needed in order to offer such services. Existing works [1, 10] focus on the case that the spot and on-demand markets are isolated from each other [9]. The pricing of spot instances is studied in [34] where a demand curve is used to describe the relation between the spot price and the number of accepted bids [34]. However, the curve is unknown in reality [5]. In [1, 10, 34], the connection of the two markets is ignored; however, the spot market is supposed to utilize the idle servers of on-demand market. Such idleness affects the capacity of spot market to accept bids and the specific schemes for assigning bids to servers.

While sharing servers among users, the model needs a feature: on-demand users can get served shortly upon arrivals, while a spot user can stably utilize a server for a properly long time once its bid is accepted. It is challenging to obtain this feature. In particular, each on-demand user requests to occupy servers in a specific period; spot users have lower priority to access servers and bid prices to utilize the unoccupied periods. After a spot user's bid is accepted and assigned to a server, the server may be preempted at any time after the assignment, since on-demand users arrive randomly. What's worse, for each accepted bid, there may be a process of loading virtual machine image to make the server ready for use; assume it takes k' minutes, e.g., k' approximates 3 [22, 29]. So, the time that the server dedicates to an accepted bid needs to be larger than k' minutes so that spot users can get some effective utilization time of servers.

We will use a discrete-time model to coordinate the assignment of all users' requests to servers; thus, time is divided into slots and each contains k minutes. These requests may arrive at any time in a slot; the assignment is performed at the beginning of the next one and it is the only action that changes the server state (i.e., occupied or idle). So, once a bid is assigned at slot t, it can stably utilize a server along the slot, without being interfered by the high priority of on-demand requests; here, the effective utilization time is $k - k'$ minutes and spot users may hope that $k - k'$ is properly large, e.g., more than ten minutes. Contradictorily, the slot duration bounds the waiting time of an on-demand request, since it needs to wait for up to k minutes to get assigned and served; so, the length k of a slot cannot be large.

Based on the observation above, we further propose a parallelized service model able to exploit the trade-off between preemption of spot instances for on-demand requests and service persistence for spot requests. In particular, all servers are divided into b groups; for the i-th group where $i \in [1, b]$, the assignment of requests occurs at the beginning of slot $t = h \cdot b + i$ where $h = 0, 1, 2, \cdots$; these requests arrive in the last slot $t - 1$. Consequently, the server state of each group can keep constant for b slots; at any slot, the arriving jobs will be connected and assigned to a specific group. So, in order to guarantee that the effective server utilization time of an accepted bid is positive, the requirement is $b \cdot k > k'$. A parallelized model allows us to set k to a value small enough and set b to a value properly large; here k can be smaller than k'. As a result, an on-demand request will get served shortly (within k minutes). After a bid is accepted, the effective server utilization time could also be large, i.e., $b \cdot k - k'$ minutes.

Main Results. Servers are shared among on-demand and spot users that arrive randomly; the former have higher priority to access servers at a fixed price. Spot users bid prices for the periods unoccupied by on-demand users. A key feature that makes such services accessible is that, on-demand users can get served within a short time upon arrivals, while a spot user can stably utilize a server for a properly long time once its bid is accepted. In this paper, we propose a discrete-time framework that has such a feature for allocating the time of servers between on-demand and spot services. The framework presents an integral process inside the system to serve the arriving on-demand and spot users: (i) assign the requests of on-demand users to servers, (ii) determine the optimal spot price, (iii) decide which spot users' bids are accepted, with the idle states of servers after the assignment of on-demand requests, and (iv) assign the requests of spot users to servers; these actions occur at the beginning of every time slot. The framework itself does not rely on any impractical assumption and can guide other CSPs to operate on-demand and spot services in their own infrastructures. Simulations show that the CSP's revenue could be improved by up to 461.5% while the resource utilization could be improved by up to 725.0%, compared with the case of only providing on-demand service.

The rest of this paper is organized as follows. In Section 2, we further explain the on-demand and spot services and introduce the related work. We propose the model and schemes for managing the requests of spot and on-demand users in Section 3; the optimal pricing of spot instances is given in Section 4. To improve quality of services, we further propose an extended framework in Section 5. Simulations are done to show the efficiency of spot pricing in Section 6. Finally, we conclude the paper in Section 7.

2 PRELIMINARIES, AND RELATED WORK

Before we formally introduce the proposed framework, we provide an overview of the state of the art on spot and on-demand services and introduce the main related works.

2.1 On-demand and Spot Services

Amazon EC2 is the current reference cloud service provider based on spot and on-demand instances. Instances are virtual machines[1] and can have different configurations of CPU, cache and disks. Instances of the same configuration have the same price and form a single market in Amazon EC2 [3]. In this paper we also consider such a homogeneous case. On-demand instances are always available at a fixed unit price and each instance's price is charged on an hourly basis. Even if partial hour of on-demand instances is consumed, the tenant will be charged the fee of the entire hour. Spot instances are with uncertain availability and their price (termed as "spot price") fluctuates over time [2]. Every user can bid a price for spot instances and they will be granted to users only if the bid price is not below the spot price. The bid price is the maximum price that the spot user can accept to pay for the spot instances. Once the spot price exceeds the user's bid price, its spot instances will get lost and terminated by Amazon EC2. Users will be charged according to the spot prices.

From a CSP's perspective, spot instances render available the computing capacity unused in the on-demand market and allow for some discount compared with the on-demand price. This permits increasing the CSP's gain in terms of revenue and user satisfiability. First, users with different delay requirements can be satisfied economically by the two types of instances [23, 39]. Latency-critical users can get service quickly by specifying a period in which to utilize instances. Delay-tolerant users can first utilize spot instances at lower prices but with uncertain availability; in case that users do not get enough instances in a long period, they can turn to on-demand instances to accelerate

[1]In this paper, we use the terms "instances", "servers", and "virtual machines" interchangeably.

processing their jobs. Next, the periods of servers unoccupied by on-demand users correspond to the idle states of on-demand instances. In the Amazon EC2 service, spot users can bid prices to utilize these states, i.e., spot instances. The pricing mechanism is not fully disclosed; however, the spot price is claimed to be set through a uniform price, sealed-bid, market-driven auction [2]: "uniform price" means all bidders pay the same price (i.e., spot price); "sealed-bid" means a user does not know the bids of the other users; "market-driven" means the spot price fluctuates based on the supply and demand of available unused EC2 capacity but is updated regularly.

2.2 Related Work

Many works have focused on characterizing spot prices over time and understanding the spot pricing scheme [19, 20]; the most relevant are [2, 34, 46]. Agmon Ben-Yehuda *et al.* analyze the time series of spot prices in different regions and define the availability of spot instances as a function of the bid price, i.e., the probability that a user successfully gets spot instances under an arbitrary bid price [2]. The authors showed that the functional curves of different types of instances in 4 regions share the same shape. Further, they conclude that spot prices in Amazon EC2 are usually drawn from a tight, fixed range of prices and are not driven by the relation of supply and demand as claimed by Amazon EC2. The authors claim that the use of such a pricing scheme can create an impression of false activity (demand and supply changes) and mask times of low demand and price inactivity, thus possibly driving up the CSP's stock.

Wang *et al.* study the optimal pricing of spot instances [34]. They assume perfect knowledge of the demand curve describing the relation of demand and price at every slot t, i.e., the number N_t of bids accepted and served under every possible spot price. Further, Lyapunov optimization is applied to derive the optimal price of spot instances by assuming that the total number L_t of bids is kept finite at every slot t; here the L_t bids contain the bids that both newly arrive at t and all bids that arrived at the previous slots but have not been served so far. However, the demand curve is unknown in reality [5]; an additional complication is that the current cloud market is still rapidly growing and unstable [13].

Zheng *et al.* derive the cost-optimal bid price for users to utilize spot instances, based on an estimated distribution of the past spot prices in Amazon EC2 [46]. In particular, similar to [34], Lyapunov optimization is used to derive the relation between the number of bid arrivals Λ_t and the spot price π_t at every slot t; as a result, by assuming the bid arrivals follow a simple distribution (e.g., exponential), the more complex distribution of spot prices could be approximated analytically. Other assumptions adopted include (i) the users' bids at every slot t follow a uniform distribution over $[\underline{\pi}, \overline{\pi}]$, and (ii) at the end of each slot, the proportion of the accepted bids that are finished is a constant. The first assumption enables simply deriving the expected number of the bids accepted at t, i.e., the fraction of bids whose prices are not below the spot price.

In [34, 46], due to their assumptions, the number of bids accepted at every slot does not need to rely on the idle state of the on-demand market. However, the spot market's capacity to accept bids is supposed to be determined by such idleness. Their models cannot account for the actual preemption scheme used for sharing server time among on-demand and spot users. Furthermore, in order to apply the Lyapunov optimization technique, an underlying assumption in [34, 46] is that all bids submitted by users will be finally accepted and served. However, this is not what Amazon EC2 promises to its spot users, and it only provides best-effort services.

Abhishek *et al.* and Dierks *et al.* analyze the performance of a hybrid spot and on-demand market using queuing theory and game theory in continuous time [1, 10]. More recently, in [10], the servers are separated into two parts, respectively serving on-demand and spot jobs. Jobs of users have diverse values and sensitivities to delay. On-demand market has a higher price but guarantees a negligible delay; in spot market, the lower a user's bid, the larger the delay of completing its jobs.

Table 1. Key Notation

Symbols	Meaning
m	the total number of servers/instances
M_t	the number of servers idle at slot t for spot market
$\mathcal{A}_t$	the set of all bids (or spot jobs) at t where $\mathcal{A}_t = \mathcal{J}_t' \cup \mathcal{J}_t''$
π_t	the spot price at t
N_t	the number of bids accepted at t
$\mathcal{J}_t'$	the bids of such users who bid successfully at $t-1$ and continues bidding at t
$\mathcal{J}_t''$	the bids that are newly submitted in the period of $[(t-1)\cdot k, t\cdot k)$
$\hat{\mathcal{J}}_t'$	the bids in $\mathcal{J}_t'$ that are also accepted at t
$\hat{\mathcal{J}}_t''$	the bids in $\mathcal{J}_t''$ accepted at t
k	a slot contains k minutes
k'	the time spent on loading/migrating virtual machine images (VMIs) is k' minutes
β	k'/k
$\hat{\mathcal{J}}_{t,1}'$	all bids of $\hat{\mathcal{J}}_t'$ that cause the operation of migrating VMI at the beginning of t
$\hat{\mathcal{J}}_{t,2}'$	$\hat{\mathcal{J}}_t' - \hat{\mathcal{J}}_{t,1}'$, in which each bid is assigned to the same server at $t-1$ and t
f_t	$\lvert\hat{\mathcal{J}}_t'' \cup \hat{\mathcal{J}}_{t,1}'\rvert$, i.e., the total number of the operations of loading/migrating VMI

Users aim to maximize their surplus and make a choice on which type of instances to use. The authors show that offering a spot market can increase the profit of a CSP. In spite of the technical merits of [1, 10], an issue in these works is that in the proposed schemes the idle servers in the on-demand market cannot be sold as spot instances, whereas making use of such idle instances is one of the main attractions of the spot market [9].

Furthermore, inspired by the dynamic pricing of Amazon EC2, there are also many works that apply the auction and mechanism design theory to cloud pricing [12, 30–33, 36, 40–42, 44, 47, 48]; the most relevant ones are [4, 16, 38]. In those frameworks, jobs truthfully report their values and latency requirements to the CSP; the CSP chooses a subset of jobs to maximize the social welfare, and process the chosen jobs on multiple machines with its capacity constraint. Unlike the model of this paper, Wu and De Pellegrini [37] analyze the performance of a type of QoS-differentiated pricing in cloud computing via an analytical approach. The CSP offers multiple QoS classes: the jobs of each class will be completed with a finite waiting time. Also, the smaller the waiting time, the higher the unit price. The CSP's servers are divided into several groups, each processing the jobs of the same class. The authors give the optimal price, the optimal rate of accepting jobs, and the minimal number of servers needed for each QoS class, thus deriving the performance of the whole system.

3 MANAGING ON-DEMAND AND SPOT SERVICES

In this section, we propose a baseline model for colocating on-demand and spot jobs on servers. The key notation is summarized in Table 1. The model refers to several schemes to (i) assign on-demand jobs to servers where every server will dedicate a specified period to the assigned job, (ii) decide which bids to accept under an arbitrary spot price, and (iii) assign spot jobs to servers. As discussed in Section 2.1, we assume that there are m homogeneous servers.

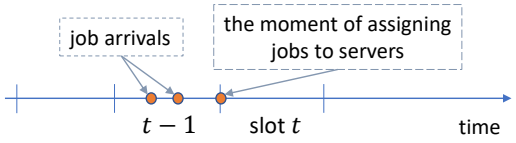

Fig. 1. Discrete-time service model where $t = 1, 2, \cdots$.

3.1 Model for Colocating On-demand and Spot Jobs

On-demand jobs have higher priority than spot jobs to access servers. On demand jobs arrive randomly and spot jobs may be preempted arbitrarily. However, the resource sharing model needs to accommodate two seemingly conflicting requirements:

> (i) Immediacy of on-demand service.] Upon arrival of an on-demand job, it can get served in a short time;
> (ii) Persistence of spot service.] After a spot job enters service, it will not be preempted shortly by on-demand jobs; ideally, the colocation scheme needs to guarantee a minimum execution time for spot jobs.

We first need to provide a discrete-time model able to describe the assignment of jobs to servers and express the above tradeoff. To this aim, time is divided into slots and the assignment occurs at the beginning of a slot. The slot duration is k minutes, and the t-th slot corresponds to the period of $[t \cdot k, (t+1) \cdot k)$ where $t = 0, 1, 2, \cdots$.

From the time point 0, jobs begin to arrive; the initial slot is slot 0 during which all servers are idle. All jobs that arrive in the period of slot $t-1$ will be available at the beginning of slot t where $t = 1, 2, \cdots$. Among these jobs, the accepted jobs will get served and dispatched to servers at the beginning of slot t. For convenience, we will simply say from a system administrator's perspective that the arrival time of these jobs is t. The discrete-time model is also illustrated in Fig. 1. In such a model, the action that changes the states of servers is the assignment of jobs to servers and it only occurs at the beginning of every slot, and their states remain constant in the period of each slot. As a result, if a spot job is executed on some server, it will not be preempted by the high priority of on-demand jobs during a slot and the minimum time that the server dedicates to it is k minutes. Formally, the discrete-time model provides the on-demand and spot services with the following properties that respectively quantify the immediacy of on-demand service and the persistence of spot service.

Feature 3.1. *On-demand users can be latency-critical. Upon arrival, they need to wait for up to k minutes to get served.*

Feature 3.2. *Spot users are usually delay-tolerant. Once bid successfully at a slot and assigned to some servers, it is guaranteed that they can persistently get served for k minutes.*

Only after providing the model for sharing servers, we shall be able to figure out the framework for assigning jobs and accepting bids. The conceptual framework is illustrated in Fig. 2 and will be elaborated in the rest of this section. In this paper, although all actions of assigning/dispatching, accepting, and pricing jobs occur at the beginning of every slot, we will simply say that they occur at slot t for convenience of exposition.

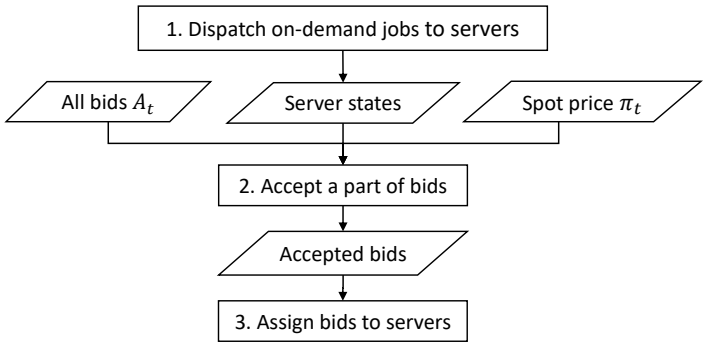

Fig. 2. A flowchart for assigning jobs and accepting bids at slot t where $t = 1, 2, \cdots$: a rectangle represents a process that executes some operations while a parallelogram represents the input or output of a process.

3.2 Dispatching High Priority of On-demand Jobs

Each on-demand user requests a time slot interval $[a_j, d_j]$ in which it can occupy a server to execute its workload where a_j and d_j are positive integers. We refer to such a request as an on-demand job j, and its arrival time, deadline and size are a_j, d_j, and $s_j = d_j - a_j + 1$ respectively. At every slot t, on-demand jobs with $a_j = t$ are dispatched to one of the m servers under some policy. Generally, any type of dispatching policies can be applied here. As summarized in [17], in IaaS services, examples of the commonly used policies include (i) *Random*: for every job j, choose one of the m servers with the probability $\frac{1}{m}$ and assign it to this server [28, 45], (ii) *Round-Robin* (RR): jobs are assigned to servers in a cyclical fashion with the j-th job being assigned to the l-th server where $l = j \bmod m$ [35], (iii) *Power of Two Choices* (PTC): for every job j, randomly choose two servers, probe them, and, assign it to the server with less queued jobs [24, 27]. The RR and Random policies are simple to implement and have similar performance; they are also supported by Amazon EC2 while serving on-demand jobs [7]. The PTC policy is more advanced but recently has been applied to the cluster management practice [27]; it can achieve a higher utilization of servers [24]. More discussion on dispatching policies can be found in [14, 15, 18, 25]. Once the job j is dispatched to a server, the server will be occupied by the on-demand job owner during the period $[a_j, d_j]$, i.e., from the beginning of slot a_j until the end of slot d_j.

On-demand instances are charged a fixed price; their users can be delay-sensitive and have no willingness to tolerate queuing delay: the current practice to guarantee quick delivery of on-demand instances to users is overprovisioning servers for on-demand jobs. As a result, while processing on-demand jobs, many servers actually remain unoccupied in the long run. The idleness in the on-demand market will be shown in Section 6.2 through experiments and available theoretical results, e.g., the load of the on-demand market corresponds to more than 85% servers in idle mode. After the job assignment at t, a server is either occupied by an on-demand job or idle in the entire period of slot t. We use M_t (*resp.* $\overline{M}_t$) to denote the number of idle (*resp.* occupied) servers in the period of t, where $\overline{M}_t + M_t = m$. The idleness of on-demand market brings the necessity of introducing spot instances into cloud market [9]; in particular, to be economically efficient, what we can do at every slot t is as follows:

- sell the idle states of on-demand market in the period of slot t in the form of spot instances; they will be accessed by spot users via bidding, and their amount is M_t.

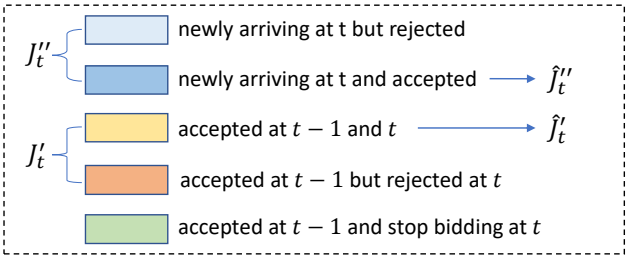

Fig. 3. Color chart for the classification of spot jobs/bids available at t when $t \geq 2$: we note that, when $t = 1$, there are only bids of $\mathcal{J}_t''$ and no other bids, e.g., $\mathcal{J}_t' = \emptyset$.

- however, on-demand jobs have higher priority to access servers, i.e., once new on-demand jobs arrive at slot $t + 1$, the instances assigned to spot users at t may be preempted arbitrarily at the beginning of $t + 1$ to serve on-demand jobs if necessary.

The additional sales of spot instances are supposed to improve the overall resource efficiency of cloud market, in contrast to a pure on-demand market.

3.3 Admission Control of Spot Jobs via Pricing

At every slot t where $t = 1, 2, \cdots$, there are A_t users who bid prices to utilize spot instances in the period of slot t; they are usually delay-tolerant. The set of these users' bids is denoted by $\mathcal{A}_t$. We assume without loss of generality that each user bids for one spot instance. Once the bid[2] of a user is accepted, it gets allocated one spot instance at t. Let π_t denote the spot price at t. Following the spirit of spot pricing described in Section 2.1, π_t is a control parameter: only the bids whose prices are not below π_t are accepted. The number of idle on-demand instances M_t defines the capacity of spot market, i.e., the number of bids accepted at t cannot exceed M_t.

Before defining a scheme for accepting bids, we first classify the bids available at t from a system administrator's perspective. This will help us describe which bids are accepted and understand how to assign these bids to servers in the next subsection. A physical spot user may bid once or several times in order to utilize spot instances at one or several slots; every time, its bid may be accepted or rejected. It stops bidding when either its entire job is completed or it wants to use other resources (e.g., on-demand instances) to complete its job. If a spot user's bid is accepted at several consecutive slots, its spot job may be assigned to and executed on the same server in this period; however, once its bid is rejected at a slot and accepted at a later slot t, it can be viewed as a user that newly arrive at t since it is not associated with and can be assigned to any server at t.

Thus, we will classify the bids of spot users at t according to their bidding behaviour at the adjacent slot $t - 1$, and there are two types of bids where $\mathcal{A}_t = \mathcal{J}_t' \cup \mathcal{J}_t''$:

- (i) $\mathcal{J}_t'$: the bids of the spot users whose bids were also accepted at slot $t - 1$;
- (ii) $\mathcal{J}_t''$: the bids of the spot users who newly arrive in the period of $[(t-1) \cdot k, t \cdot k)$.

Specially, if $t = 1$, $\mathcal{J}_t' = \emptyset$ since there are no bids accepted at the initial slot $t - 1 = 0$ during which there are newly arriving bids alone. For each bid of $\mathcal{J}_t'$, its owner also bids at $t - 1$; as seen later, at $t - 1$ and t, the two bids will be assigned to the same server when there is no interference from the high priority of on-demand jobs. Finally, $\mathcal{A}_t$ denotes all bids available at t: $\mathcal{J}_t'$ is illustrated by the orange and gold rectangles in Fig. 3; $\mathcal{J}_t''$ is illustrated by the light and heavy blue rectangles.

[2]In this paper, each bid at a slot corresponds to a spot job who aims to utilize an instance for one slot once accepted; we shall use the terms "bids" and "spot jobs" interchangeably.

Algorithm 1: Determination of bids accepted at t

```
if |F(π_t, A_t)| ≤ M_t then
    accept |F(π_t, A_t)| bids of A_t with the highest bid prices;
else
    accept M_t bids of A_t with the highest bid prices;
```

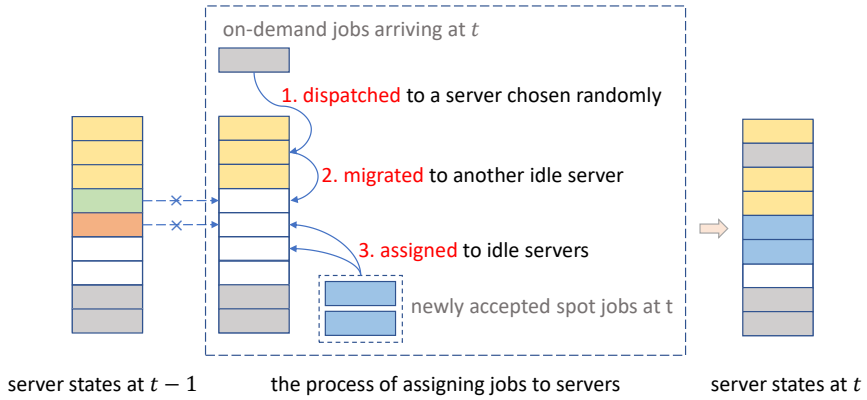

Fig. 4. Assignment of on-demand and spot jobs to servers at the beginning of slot t when $t \geq 2$: a colored rectangle denotes either a job of some type or the state of a server (i.e., a server occupied by this type of jobs); the meaning of different colors is partially summarized in Fig. 3; a blank rectangle denotes an idle server.

Acceptance of spot jobs. The decision of accepting bids is made at the beginning of slot t. To describe the bids accepted at a slot, we define a function as follows. Generally, let $\mathcal{J} = \{1, 2, \cdots, J\}$ denote a set of users who submit bids where $J = |\mathcal{J}|$, and $\mathcal{V} = \{v_1, v_2, \cdots, v_J\}$ denote the set of their bid prices where $v_1 \geq v_2 \geq \cdots \geq v_J$. Let α denote a non-negative real number; we define a function as follows:

$$F(\alpha, \mathcal{J}) = \{i \,|\, v_i \geq \alpha, i \in \mathcal{J}\}. \tag{1}$$

$F(\alpha, \mathcal{J})$ denotes all users whose bid prices are no smaller than α. The number of the bids whose prices are not below π_t is $|F(\pi_t, \mathcal{A}_t)|$. The procedure for determining which bids are accepted is presented in Algorithm 1; at slot t, the number of accepted bids is the minimum of M_t and $|F(\pi_t, \mathcal{A}_t)|$, i.e.,

$$N_t = \min\{M_t, |F(\pi_t, \mathcal{A}_t)|\}. \tag{2}$$

Also, we note that M_t and $\mathcal{A}_t$ are observable at the beginning of slot t and thus known by CSP. We denote by $\hat{\mathcal{J}}_t'$ (resp. $\hat{\mathcal{J}}_t''$) the accepted bids of $\mathcal{J}_t'$ (resp. $\mathcal{J}_t''$). Here, $\hat{\mathcal{J}}_t'$ and $\hat{\mathcal{J}}_t''$ are illustrated by gold and heavy blue rectangles in Fig. 3. In the rest of this section we shall refer to the color chart of Fig. 3 to support the description of job assignment with a graphical representation.

3.4 Assignment of Spot Jobs to Servers

At every slot $t = 1, 2, \cdots$, all the arriving on-demand jobs will be accepted but only a part of bids may be accepted. On-demand jobs have higher priority; when dispatching them to servers, the existence of spot jobs is ignored as if there is a pure on-demand market. The scheme for assigning jobs to servers is as follows:

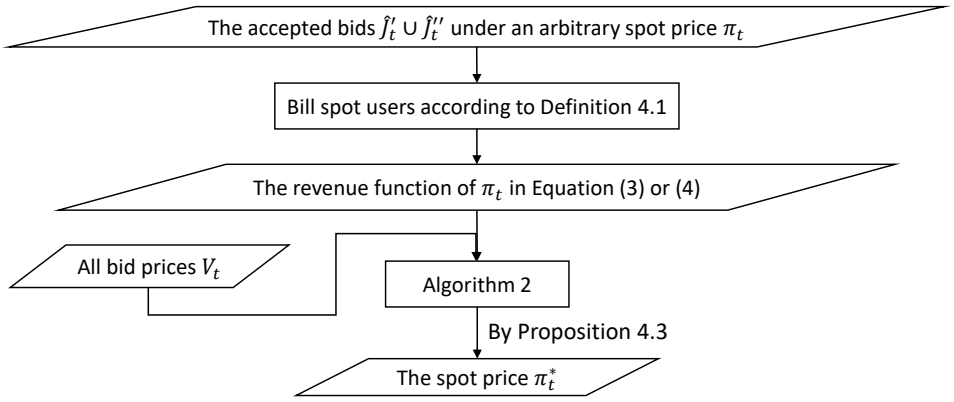

Fig. 5. The optimal pricing of spot instances at the beginning of slot t.

Step 1. On-demand jobs are dispatched to servers using some policy such as "Random", "RR" or "PTC".

Step 2. In the case that $t \geq 2$, if an on-demand job is dispatched to a server that was executing a spot job in $\hat{\mathcal{J}}_t'$ in the period of slot $t-1$, this spot job is migrated to another idle server; the other jobs in $\hat{\mathcal{J}}_t'$ are still executed on the same instances. In the case that $t = 1$, go to the next step directly since $\hat{\mathcal{J}}_t' = \emptyset$.

Step 3. the spot jobs of $\hat{\mathcal{J}}_t''$ are randomly dispatched to the remaining idle servers.

In the period of slot 0, all servers are in idle states. At the beginning of slot 1, there are only bids of $\hat{\mathcal{J}}_t''$ and no bids in $\hat{\mathcal{J}}_t'$; these bids are assigned to servers after the assignment of on-demand jobs. When $t \geq 2$, a part of servers might be occupied by on-demand and spot jobs in the period of slot $t-1$; the process of assigning jobs to servers is also illustrated in Fig. 4. We note that, when $t = 1$, the states of all servers at $t-1$ will be represented by blank rectangles and the second step in the process of assigning jobs to servers could be removed since nothing is executed.

4 OPTIMAL PRICING OF SPOT INSTANCES

In this section, we shall determine the optimal spot price to maximize the CSP's revenue at every slot t. We denote it by π_t^*: it is the spot price finally announced to users at t. It actually determines the subset of bids accepted by Algorithm 1. Afterwards, we will conclude Sec. 3 and Sec. 4 by showing the whole framework for running spot and on-demand services.

As is formally shown latter, the optimal spot price may be the bid price of some user. This can be intuitively perceived by contradiction: if the value of π_t^* is between two bid prices, the revenue of a CSP will be increased by resetting π_t^* to the larger bid price, and doing so does not affect the acceptance of spot jobs. To determine which bid price can maximize the revenue, we need to characterize the CSP's revenue function under an arbitrary spot price π_t. Under π_t, the accepted bids $\hat{\mathcal{J}}_t' \cup \hat{\mathcal{J}}_t''$ are determined by the framework in Section 3 (illustrated in Fig. 2). The framework for pricing spot instances at t is illustrated in Fig. 5 and will be elaborated in the following.

4.1 Billing and Revenue

We first define the value that a CSP gets from each spot job accepted at t, i.e., the way of billing. The reference time interval for billing an on-demand user is made of L consecutive slots. The price

of utilizing an instance for one interval is p: we recall that if a user utilizes just a fraction of an interval, it is still charged the fee of the entire interval.

Conversely, for each accepted bid, we should define the billing way according to the effective utilization time of a spot instance. We observe that, although a spot instance is assigned by reserving a whole slot time, there may be yet some overhead. In fact, in some cases a process of startup/migration is needed: it takes some time during which the job is in service but inactive. The spot price at slot t is π_t, and, to be comparable with the on-demand price, π_t is the price of utilizing a spot instance for an interval of L slots. However, we shall use a proper normalization, since π_t is limited to represent the spot price of slot t. For example, if a spot user effectively utilizes an instance for one minute during the slot t, the actual charged price will be $\pi_t/(L \cdot k)$, where a slot contains k minutes.

Now, we observe the period in which servers are effectively utilized by spot jobs after the job assignment at t. Recall the job assignment process in Section 3.4. For spot jobs newly accepted $\hat{\mathcal{J}}_t''$, their virtual machine images (VMIs) need to be loaded to the assigned servers. As illustrated in Fig. 4, some spot jobs of $\hat{\mathcal{J}}_t'$ may need migration, denoted by $\hat{\mathcal{J}}_{t,1}'$, and their VMIs also need be migrated to other servers. As shown in related studies [22, 29], for state of art technology, the process of loading or migrating VMIs takes about 3 minutes; generally, we denote the time consumed for this process by k' minutes. Thus, for the jobs of $\hat{\mathcal{J}}_t'' \cup \hat{\mathcal{J}}_{t,1}'$, although the whole period of a slot will be dedicated to them, only $k - k'$ minutes are effectively utilized, with k' minutes not utilized for actual service. For the other spot jobs of $\hat{\mathcal{J}}_t'$, they were ever executed on some servers at $t-1$ and will still be executed on the same servers at t; we denote these bids by $\hat{\mathcal{J}}_{t,2}'$ where $\hat{\mathcal{J}}_{t,2}' = \hat{\mathcal{J}}_t' - \hat{\mathcal{J}}_{t,1}'$. The bids of $\hat{\mathcal{J}}_{t,2}'$ can effectively utilize the whole slot t. In order to guarantee that the effective server utilization time of every accepted bid is positive, the following relation needs to be satisfied:

$$k' < k.$$

Let $\beta = \frac{k'}{k}$ where $\beta \in (0, 1)$, and every spot job will be charged for the period in which the servers are effectively utilized. Thus, we have the following definition.

Definition 4.1. The way of billing a spot user at t is as follows: (i) every accepted bid in $\hat{\mathcal{J}}_t'' \cup \hat{\mathcal{J}}_{t,1}'$ is charged $(1-\beta) \cdot \frac{\pi_t}{L}$, and (ii) every accepted bid in $\hat{\mathcal{J}}_{t,2}'$ is charged $\frac{\pi_t}{L}$.

Now, we characterize the revenue function and show which system information is observable at the beginning of t. Given any spot price π_t, the accepted bids (i.e., $\hat{\mathcal{J}}_t''$, $\hat{\mathcal{J}}_{t,1}'$, $\hat{\mathcal{J}}_{t,2}'$) are determined by Algorithm 1, before which on-demand jobs have been assigned to servers. Also, we know the locations of the servers to which the bids accepted at $t-1$ are assigned, as illustrated in Fig. 4. So, after the bids accepted at t are determined, we could know the bids respectively in $\hat{\mathcal{J}}_{t,1}'$, $\hat{\mathcal{J}}_{t,2}'$ and $\hat{\mathcal{J}}_t''$. Let f_t denote the total number of the accepted bids of $\hat{\mathcal{J}}_t'' \cup \hat{\mathcal{J}}_{t,1}'$; f_t is also observable after the accepted bids are determined. The total number of the bids accepted at t is N_t; so, $|\hat{\mathcal{J}}_{t,2}'| = N_t - f_t$. The revenue of the spot market at t is the sum of the charges of all accepted bids, denoted by $\mathcal{G}(t)$; with the billing policy in Definition 4.1, we have

$$\mathcal{G}(t) = (N_t - f_t) \cdot \frac{\pi_t}{L} + f_t \cdot (1-\beta) \cdot \frac{\pi_t}{L} = (N_t - \beta \cdot f_t) \cdot \frac{\pi_t}{L} \tag{3}$$

where N_t is given in (2), L and β are system parameters, and f_t is observable.

4.2 Pricing Decision

Our decision-making problem is determining the optimal π_t to maximize the spot market's revenue $\mathcal{G}(t)$ at slot t. By (2), N_t is a function of π_t, $\mathcal{A}_t$ and M_t; the revenue function $\mathcal{G}(t)$ in (3)

Algorithm 2: SpotiPrice(M_t, $\mathcal{A}_t$, f_t, L, β)

$\pi' \leftarrow 0$, $G' \leftarrow 0$;
for $i \leftarrow 0$ **to** A_t **do**
  $N_t \leftarrow \min\{M_t, |\mathcal{F}(v_i, \mathcal{A}_t)|\}$;
  $G \leftarrow N_t \cdot \frac{v_i}{L} - \beta \cdot f_t \cdot \frac{v_i}{L}$; // the revenue of spot market if the spot price at t is v_i
  if $G' < G$ **then**
    $G' \leftarrow G$, $\pi' \leftarrow v_i$;
$\pi_t^* \leftarrow \pi'$; // the optimal spot price at slot t

can be expressed as a function of the single variable π_t, i.e.,

$$\mathcal{G}(t) = \hat{\mathcal{G}}\,(\pi_t,\, M_t,\, \mathcal{A}_t,\, f_t) \tag{4}$$

where parameters M_t, $\mathcal{A}_t$ and f_t are observable at slot t. Sort the spot jobs of $\mathcal{A}_t$ in the non-increasing order of their bid prices; let v_j denote the bid price of the j-th spot job where

$$v_1 \geq v_2 \geq \cdots \geq v_{A_t}, \tag{5}$$

where $A_t = |\mathcal{A}_t|$. Let $v_0 = v_1 + 1 > v_1$, and $\mathcal{V}_t = \{v_0, v_1, \cdots, v_{A_t}\}$; then, we draw the following conclusion.

Lemma 4.2. *In order to maximize the revenue of spot market at slot t, the optimal spot price is such that $\pi_t^* \in \mathcal{V}_t$. Let N_t^* denote the number of bids accepted at t and we have $\pi_t^* = v_{N_t^*}$.*

Proof. Suppose that in an optimal solution N_t^* bids are accepted; as defined in Section 3.3, only the bids whose prices are no smaller than π_t^* are possibly accepted, and these bids are the N_t^* bids of $\mathcal{A}_t$ with the highest bid prices. If $N_t^* = 0$, no bid is accepted at t and π_t can be an arbitrary value larger than v_1; so, π_t^* can be set to v_0. If $N_t^* > 0$, the optimal spot price $\pi_t^* \leq v_{N_t^*}$. Then, the CSP's revenue function $\mathcal{G}(t)$, given in (3), is maximized when setting π_t^* to the highest possible price, i.e. $\pi_t^* = v_{N_t^*}$. The reason for this is that, when $\pi_t \in [0, v_{N_t^*}]$ and the number of accepted bids N_t is fixed and equals N_t^*, $\mathcal{G}(t)$ is an increasing function of π_t since $N_t - \beta \cdot f_t > 0$ where $N_t \geq f_t$ and $\beta \in (0, 1)$. Finally, the lemma holds. □

Proposition 4.3. *The optimal spot price π_t^* at slot t is as follows:*

$$\pi_t^* \leftarrow \arg\max_{\pi_t \in \mathcal{V}_t} \hat{\mathcal{G}}(\pi_t, M_t, \mathcal{A}_t, f_t). \tag{6}$$

Proof. The optimal spot price at t is some value in $\mathcal{V}_t$ under which $\mathcal{G}(t)$ achieves the maximal value, by Lemma 4.2; hence, the proposition holds. □

At every slot t, the expression (6) could be used to decide the optimal spot price, and the corresponding procedure is presented in Algorithm 2: it checks every possible value in $\mathcal{V}_t$ to see which can maximize the revenue function (3). *A key feature of our algorithm* is that such decisions are implementable in practice, since the CSP has full knowledge of all parameters in $\mathcal{G}(t)$ except the control parameter, i.e., the spot price π_t, at every t.

4.3 Running Spot and On-demand Services

So far, we have shown in Sec. 3 and Sec. 4 an integral framework for running spot and on-demand services. Now, we explain how this framework works as a whole.

Sharing model. The discrete-time service model is proposed in Section 3.1 for sharing server time among on-demand and spot jobs where time is divided into consecutive slots. The jobs that arrive in the period of slot $t-1$ will be assigned at the next slot t, as illustrated in Fig. 1, where $t = 1, 2, \cdots$. The action (i.e., job assignment) that changes the states of servers happens only at the beginning of each slot t and their states keep constant along the time slot. After the job assignment, an on-demand job j will utilize the server for s_j slots while an accepted spot job can stably access the server for one slot.

Job assignment, pricing and acceptance. While running on-demand and spot services, several actions are coordinated to control the job's access to servers and they occur sequentially at every slot $t = 1, 2, 3, \cdots$. At t, on-demand jobs are first dispatched to servers and then two processes happen. The first is the calculation of the optimal spot price π_t^* at t: with the framework in Section 3 (illustrated in Fig. 2), we could determine by Algorithm 1 the bids accepted at t under an arbitrary spot price π_t. Based on this, by the framework illustrated in Fig. 5, π_t^* could be derived. The second process actually determines the acceptance and assignment of spot jobs: π_t^* is the final spot price announced to users and is actually used as the input of the framework illustrated in Fig. 2; then, the actions of accepting bids and assigning spot jobs to servers happen.

We have concluded the presentation of the basic framework. Finally, we analyze the spot user's behavior from a game theory perspective. For every spot user j, the willingness-to-pay (WTP) of j is the maximum price at or below which it can accept the spot service and we denote its WTP by c_j. If the bid of j is accepted and it gets spot service at t, let ς_j denote the effective server utilization time during the slot t; let $\hat{o}_j = (c_j - \pi_t) \cdot \frac{\varsigma_j}{L}$, and the payoff of j equals $\hat{o}_j$. If its bid is rejected, the payoff equals zero. In any case, we denote by o_j the payoff of j. A spot user j will truthfully report its WTP if its payoff o_j is maximal or at least not less by being truthful, regardless of what the others do.

Proposition 4.4. *At every slot t, when a user j bids a price for spot instances, it will truthfully report its WTP to the CSP, i.e., its bid price v_j equals its WTP c_j.*

Proof. There are A_t bids whose prices satisfy (5). By Lemma 4.2 and Algorithm 1, we have in an optimal solution that the N_t bids with the highest prices will be accepted and $\pi_t = v_{N_t}$. Thus, we have that (1) all bids whose prices are larger than π_t will be accepted, (2) a bid whose price equals π_t is possibly accepted, subject to the capacity constraint, and (3) all bids whose prices are smaller π_t will be rejected. It suffices to show that the payoff of j is maximal or not less by being truthful, regardless of the value of π_t. When j misreports its WTP, there are two cases: (i) $v_j > c_j$ and (ii) $v_j < c_j$. First, we analyze the first case. (i.a) If $\pi_t > v_j$, user j is rejected with $o_j = 0$ no matter whether it is truthful. (i.b) If $\pi_t = v_j$, it may be accepted or not; we have $o_j = \hat{o}_j < 0$ if accepted and $o_j = 0$ otherwise. By being truthful, it is rejected with $o_j = 0$. (i.c) If $c_j < \pi_t < v_j$, it is accepted with $o_j = \hat{o}_j < 0$; by being truthful, it is rejected with $o_j = 0$. (i.d) If $c_j = \pi_t$, it is accepted with $o_j = 0$; by being truthful, we also have $o_j = 0$ no matter whether it is accepted. (i.e) If $\pi_t < c_j$, it is accepted and gets the same payoff o_j no matter whether it is truthful. Next, we analyze the second case in a similar way. (ii.a) If $\pi_t < v_j$, it is accepted and gets the same payoff no matter whether it is truthful. (ii.b) If $\pi_t = v_j$, it may be accepted or not with the payoff o_j equaling $\hat{o}_j$ or 0; by being truthful, it is accepted with $o_j = \hat{o}_j > 0$. (ii.c) If $v_j < \pi_t < c_j$, it is rejected with $o_j = 0$; by being truthful, it is accepted with $o_j = \hat{o}_j > 0$. (ii.d) If $\pi_t = c_j$, it is rejected with $o_j = 0$; by being truthful, it may be accepted or not with $o_j = 0$ in any case. (ii.e) If $\pi_t > c_j$, it is rejected with $o_j = 0$ no matter whether it is truthful. Observing in the analysis above, we can conclude that the payoff of j is maximal or not less by being truthful; thus, the proposition holds. □

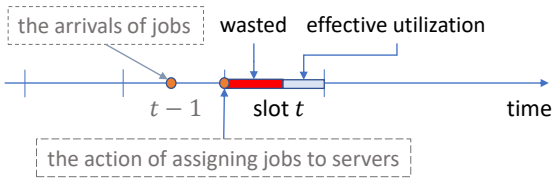

Fig. 6. The limits of usability of the basic model: the length of the red rectangle is the time used for loading for loading/migrating VMIs, where no task can be executed.

5 AN EXTENDED FRAMEWORK

In this section, we provide further insight into the basic framework introduced Sec. 3 and Sec. 4, and propose an extension attaining higher quality of service.

5.1 Limitation to Usability

In the basic framework, the slot duration has two implications as indicated in Feature 3.1 and Feature 3.2. First, since an on-demand job may arrive at any time point in the period of a slot and will get served at the beginning of the next slot, the delivery of computing service is delayed to some extent. In the worst case up to k minutes are required before being allocated. This can harm the quality of on-demand service and

- from an on-demand user's perspective, it may hope that the slot duration is not large.

Second, upon acceptance of a spot job at a slot t, a server will be allocated to this job. The slot duration represents the guaranteed time that the server will dedicate to the job. Such a spot job risks being rejected at the next slot $t+1$ since the spot price may change. Furthermore, spot jobs newly accepted or spot jobs migrated to another server to prioritize new on-demand jobs face a process of migrating or loading VMIs, which takes k' minutes, e.g., k' approximates 3 under current technology. Ultimately, such spot jobs can effectively utilize servers for $k-k'$ minutes, where $k > k'$. So, we have

- from a spot user's perspective, it may hope that the slot duration is properly large.

In the basic framework, both the persistence of spot service and the immediacy of on-demand service depend on the slot duration; it is difficult to simultaneously satisfy the requirements of on-demand and spot users. In fact, in Amazon EC2, k is set to 5; in this case, upon acceptance of a bid at t, 3 minutes are wasted while 2 minutes are effectively utilised, as illustrated in Fig. 6; however, an on-demand user may need to wait for up to 5 minutes to get served. Finally, we observe that the process of loading or migrating VMIs requires additional system resources (e.g., bandwidth) to be consumed. From a system administrator's perspective, the convenience of the spot-pricing scheme may be reduced if it generates a large number of such operations.

5.2 Our Improvement: Implementing Spot Pricing in Parallel

In this subsection, we propose an extended framework to solve the tradeoff between the persistence of spot service and the immediacy of on-demand service; here, the basic framework of Sec. 3 and Sec. 4 will be implemented in parallel on multiple groups of servers that alternatingly serve the jobs that arrive at different slots.

Parallelized model. There are a total of m servers. In the basic framework, every job accepted at t will be assigned to one of the m servers where $t = 1, 2, 3, \cdots$. Now, the servers are divided into b groups, and the i-th group consists of m_i servers where $\sum_{i=1}^{b} m_i = m$. For all on-demand and spot jobs accepted at any slot t, there exists a $i \in [1, b]$ such that all these jobs will be assigned to the

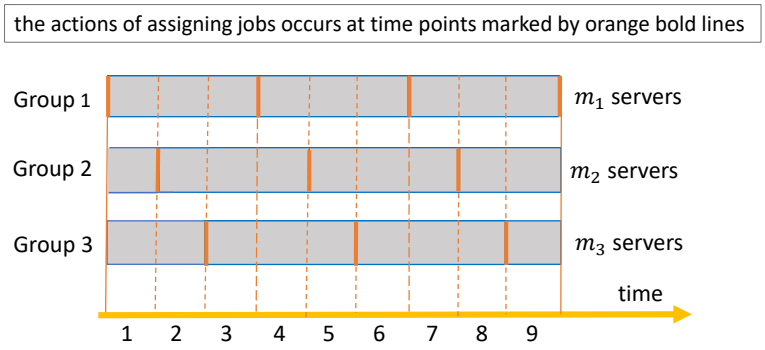

Fig. 7. An extended framework: for all $i \in [1, 3]$, the i-th group serves the jobs arriving and accepted at slot $3 \cdot h + i$ where $h = 0, 1, 2, \cdots$.

servers of the i-th group; here, t and i satisfy the following relation:

$$h = \left\lceil \frac{t}{b} \right\rceil - 1 \text{ and } i = t - h \cdot b, \tag{7}$$

where b is a system parameter; for example, if $b = 2$, the jobs arriving at slot $t = 1, 3, 5, \cdots$ will be served by the first group of servers. In other words, for all $i \in [1, b]$, the i-th group is an independent processing unit that serves the on-demand and spot jobs that arrive at slot $t = h \cdot b + i$ (i.e., in the period of slot $t - 1$) where $h = 0, 1, 2, \cdots$; these jobs arrive every b slots. Within any group, the schemes for processing jobs are similar to the ones in the basic framework, which will be elaborated later.

At the i-th group, the action of assigning jobs to servers only occurs at the beginning of slot $t = h \cdot b + i$ where $h = 0, 1, 2, \cdots$. The key observation is that only such actions will change the server states, and the server states of the i-th group keep constant for b slots, i.e., $b \cdot k$ minutes. Now, we have two parameters k and b to control the jobs' waiting and service time. After the assignment of jobs at t, the spot jobs can utilize the assigned server for b slots without being interrupted by the high priority of on-demand jobs that arrive at the subsequent $b - 1$ slots; those latter on-demand jobs will be served by the other $b - 1$ groups of servers. Such an extended framework is also illustrated in Fig. 7 where $b = 3$, and it has the following two features that quantify the immediacy of on-demand service and the persistence of spot service:

Feature 5.1. *Upon arrival, on-demand users need to wait for up to k minutes to get served.*

Feature 5.2. *Once spot users bid successfully, it is guaranteed that they can persistently get served for $b \cdot k$ minutes.*

Parameter Setting. In this framework, the parameters k', b, k, and L are set to satisfy the following relations: (i) $k' < b \cdot k$, and (ii) L is the multiple of b, i.e., $K = \frac{L}{b}$ is an integer. The first relation guarantees for each accepted bid that the effective server utilization time is positive, i.e., $b \cdot k - k' > 0$; here, the value of k can be smaller than k'. Instead, to guarantee this, it is required that $k > k'$ in the basic framework of Sec. 3 and Sec. 4. Thus, in the extended framework, we can set the length of a slot to a small value, which can well guarantee the immediacy of on-demand service by Feature 5.1. For example, we set $k = 1$ (minute); then, an on-demand job needs to wait for at most 1 minute to get served. On the other hand, we can set b to a large value such that $b \cdot k > k'$, which can well guarantee the persistence of spot service by Feature 5.2. For example, when $b = 30$, the minimum time that a server dedicates to an accepted bid is up to 30 minutes. As a result, with

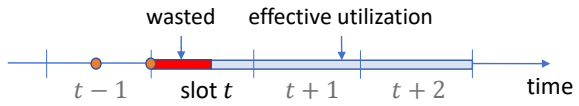

Fig. 8. Improvement to Usability.

the extended framework, we can well address the two seemingly conflicting requirements from spot and on-demand users above. We also illustrate in Fig. 8 another case where $k = 5$, $b = 3$, and $k' = 3$, in contrast to Fig. 6; here, upon acceptance of a bid, 3 minutes are wasted and 12 minutes are effectively utilized.

At the i-th group, on-demand jobs arrive at $t = h \cdot b + i$ where $h = 0, 1, 2, \cdots$; they have higher priority to access servers and are first dispatched to the servers. Recall that the reference time interval for billing an on-demand user is made of L consecutive slots. Thus, the size s_j of an on-demand job j can be viewed as the multiple of L (slots) and let $\tau_j = \frac{s_j}{L}$. After the assignment of j at slot t, it will occupy a server for $s_j = b \cdot K \cdot \tau_j$ slots. The second relation guarantees that the number of slots occupied by j is the multiple of b. Thus, for all $t' = h' \cdot b + i$ where $h' = 0, 1, 2, \cdots$, if the slot t' of a server is occupied by an on-demand job, the entire period of $[t', t' + b - 1]$ will be occupied by this job.

Job assignment, acceptance and pricing. For all $i \in [1, b]$, the i-th processing unit is a single system where the way of processing jobs is the same as the way of the basic framework, except that there are only m_i servers available and jobs arrive every b slots, i.e., at $t = h \cdot b + i$ where $h = 0, 1, 2, \cdots$. In the following, we apply the notion in the basic framework to the scenario here and show the process of assigning and accepting jobs, as described in Section 3.2, 3.3 and 3.4 and illustrated in Fig. 2; then, we explain the process of determining the optimal spot price at t, as described in Section 4 and illustrated in Fig. 5.

After the assignment of on-demand jobs, the idle servers of the i-th unit in the period of $[t, t+b-1]$ are sold as spot instances where $t = h \cdot b + i$ and $h = 0, 1, 2, \cdots$. We denote their amount by $M_t^{(i)}$, which is also the capacity of accepting bids at t. Spot users arrive at every such slot t and bid prices to utilize spot instances. All bids available at t are denoted by $\mathcal{A}_t = \mathcal{J}'_{i,t} \cup \mathcal{J}''_{i,t}$: $\mathcal{J}'_{i,t}$ denotes the bids that belongs to the spot users whose bids have been accepted at $t - b$ and who continue bidding at t, and $\mathcal{J}''_{i,t}$ denotes the bids newly arriving and submitted at any time point in the period of slot $t - 1$. Replacing the $\mathcal{J}'_t$, $\mathcal{J}''_t$ and M_t of Algorithm 1 with $\mathcal{J}'_{i,t}$, $\mathcal{J}''_{i,t}$ and $M_t^{(i)}$, we could get the procedure for accepting bids in this section. As is given in (2), the number of accepted bids at t depends on the spot price π_t, the available bids $\mathcal{A}_t$ and the number of spot instance $M_t^{(i)}$ and is

$$N_t^{(i)} = \min\left\{M_t^{(i)}, |F(\pi_t, \mathcal{A}_t)|\right\}. \tag{8}$$

Let $\hat{\mathcal{J}}'_{i,t}$ denote the accepted bids in $\mathcal{J}'_{i,t}$, and $\hat{\mathcal{J}}''_{i,t}$ denote the accepted bids in $\mathcal{J}''_{i,t}$. The procedure of assigning spot jobs to servers is the same as the procedure in Section 3.4 after (i) we replace the accepted bids $\hat{\mathcal{J}}'_t$ and $\hat{\mathcal{J}}''_t$ with $\hat{\mathcal{J}}'_{i,t}$ and $\hat{\mathcal{J}}''_{i,t}$, and (ii) we replace the period of slot $t - 1$ in the Step 2 with the period of $[t - b, t - 1]$. Once an instance is offered to a spot job, the instance will dedicate b time slots to this job.

For the accepted bids where the operation of loading or migrating VMIs is needed, their amount is denoted by $f_t^{(i)}$, which is observable. Recall that $\beta = \frac{k'}{k}$, and there are $f_t^{(i)}$ accepted bids whose effective server utilization time is $b \cdot k - k'$ minutes, i.e., $b - \beta$ slots; for the other accepted bids, the effective utilization time is b slots and their amount is $N_t^{(i)} - f_t^{(i)}$. As stated in Definition 4.1, the instances will be charged for the period in which they are effectively utilized for executing

workload, excluding the period in which VMIs are loaded or migrated; the price of effectively utilizing an instance for a slot is $\frac{\pi_t}{L}$. Similar to (3), the revenue from the spot market at t is as follows:

$$\mathcal{G}^{(i)}(t) = \left(N_t^{(i)} - f_t^{(i)}\right) \cdot \frac{\pi_t}{K} + f_t^{(i)} \cdot \left(1 - \frac{\beta}{b}\right) \cdot \frac{\pi_t}{K} = \left(N_t^{(i)} - \frac{\beta}{b} \cdot f_t^{(i)}\right) \cdot \frac{\pi_t}{K}, \tag{9}$$

where $K = \frac{L}{b}$. With (8), $\mathcal{G}^{(i)}(t)$ can be transformed as a function of the single variable π_t:

$$\mathcal{G}^{(i)}(t) = \check{\mathcal{G}}\left(\pi_t,\, M_t^{(i)},\, \mathcal{A}_t,\, f_t^{(i)}\right), \tag{10}$$

where $M_t^{(i)}$, $\mathcal{A}_t$ and $f_t^{(i)}$ are observable at slot t. As we conclude in Lemma 4.2 and Proposition 4.3, the optimal spot price at t is in $\mathcal{V}_t$ and is such that

$$\pi_t^* \leftarrow \arg\max_{\pi_t \in \mathcal{V}_t} \check{\mathcal{G}}\left(\pi_t,\, M_t^{(i)},\, \mathcal{A}_t,\, f_t^{(i)}\right); \tag{11}$$

the corresponding procedure is presented by Algorithm 2, i.e., SpotiPrice$\left(M_t^{(i)}, \mathcal{A}_t, f_t^{(i)}, K, \beta/b\right)$.

Reference Performance Metric. After giving the framework for running on-demand and spot services, a further objective of this paper is evaluating its performance. In the extended framework, the states of all servers still keep constant in the period of every slot; for all $l \in [1, b]$, we denote by $\overline{M}_t^{(l)}$ the number of servers of the l-th processing unit used as on-demand instances at slot t. Let $\overline{M}_t = \sum_{l=1}^{b} \overline{M}_t^{(l)}$, denoting the total number of instances that are occupied by on-demand jobs at t. Recall that the price of utilizing an on-demand instance for L slots is p, and at slot t the revenue from the on-demand market is

$$\mathcal{G}_t^o = \frac{p}{L} \cdot \overline{M}_t. \tag{12}$$

In contrast, the revenue from the spot market at t is $\mathcal{G}^{(i)}(t)$ given in (9). The revenue improvement brought by the spot market is measured by the following ratio:

$$\alpha_t = \frac{\mathcal{G}^{(i)}(t)}{\mathcal{G}_t^o}. \tag{13}$$

The ratio α_t represents how much the CSP's revenue could be improved by at slot t after complementing the on-demand market with a spot market. The revenue improvement α_t is a main performance metric of this paper.

5.3 An Efficiency Analysis

The extended framework of this section allows us to use some assumptions to derive an analytical result. The aim is to characterize the revenue improvement α_t with a simple mathematical expression; it helps us clearly understand which factors are affecting the revenue improvement when the on-demand market is complemented with a spot market via the framework proposed in this paper. The two assumptions are: (i) the product $b \cdot k$ is set to a large enough value, and (ii) at every slot t, the prices of all bids $\mathcal{A}_t$ follow a uniform distribution over $[\underline{\pi}, \overline{\pi}]$, as is used for cloud services in [46]; here, $\overline{\pi}$ and $\underline{\pi}$ are the maximum and minimum bid prices of users, and $\overline{\pi}$ can be viewed as the on-demand price p, i.e., $\overline{\pi} = p$. We emphasize that our framework itself do not rely on such assumptions in order to run on-demand and spot services.

Under the first assumption we have $\frac{k'}{b \cdot k} = \frac{\beta}{b} \to 0$ where k' is a fixed parameter; furthermore, $f_t^{(i)}/N_t^{(i)} \le 1$. Thus, by (9), the revenue from the spot market at every slot t can be approximated as

$$\mathcal{G}^{(i)}(t) = N_t^{(i)} \cdot \frac{\pi_t}{K}. \tag{14}$$

Under the second assumption we have that the expected number of bids whose prices are not below π_t is $|F(\pi_t, \mathcal{A}_t)| = A_t \cdot (\overline{\pi} - \pi_t)/(\overline{\pi} - \underline{\pi})$. The number of accepted bids given in (8) can be transformed as

$$N_t^{(i)} = |F(\pi_t, \mathcal{A}_t)| = A_t \cdot \frac{\overline{\pi} - \pi_t}{\overline{\pi} - \underline{\pi}}, \tag{15}$$

subject to the constraint that $N_t^{(i)} \leq M_t^{(i)}$. Let

$$\pi_t' = \overline{\pi} - \frac{M_t^{(i)}}{A_t} \cdot \left(\overline{\pi} - \underline{\pi}\right), \text{ and } \pi_t'' = \max\left\{\pi_t', \underline{\pi}\right\}. \tag{16}$$

With (15), the constraint translates to $\pi_t \geq \pi_t'$. Since $\pi_t \in [\underline{\pi}, \overline{\pi}]$, the spot price at t should satisfy

$$\pi_t \geq \pi_t''. \tag{17}$$

Finally, at any slot $t = h \cdot b + i$, we have from (14) and (15) that, the CSP's revenue from the spot market is as follows

$$\mathcal{G}^{(i)}(t) = \frac{A_t}{K \cdot \left(\overline{\pi} - \underline{\pi}\right)} \cdot \left(\overline{\pi} \cdot \pi_t - \pi_t^2\right). \tag{18}$$

At t, A_t is observable; $\mathcal{G}^{(i)}(t)$ is a quadratic function of π_t subject to (17). We let $\rho = \underline{\pi}/\overline{\pi}$ and have that

Proposition 5.1. *At every slot t, the optimal spot price π_t^* and the maximum revenue from the spot market are as follows:*

$$\mathcal{G}^{(i)}(t) = \begin{cases} \dfrac{\overline{\pi}}{4 \cdot K} \cdot \dfrac{A_t}{1-\rho} & \text{if } \rho \leq \min\{0.5, 1 - D/2\}, \text{ where } \pi_t^* = \overline{\pi}/2 \\ A_t \cdot \dfrac{\underline{\pi}}{K} & \text{if } D \leq 1 \text{ and } \rho > 0.5, \text{ where } \pi_t^* = \underline{\pi} \\ \dfrac{\overline{\pi}}{K} \cdot (1 - \dfrac{1-\rho}{D}) \cdot M_t^{(i)} & \text{if } D > 1 \text{ and } \rho > 1 - D/2, \text{ where } \pi_t^* = \pi_t'. \end{cases} \tag{19}$$

where π_t' is given in (16), $D = \frac{A_t}{M_t^{(i)}}$, and $\rho \in (0, 1)$.

Proof. For the quadratic function $\mathcal{G}^{(i)}(t)$, the axis of symmetry is a vertical line $x = \frac{\overline{\pi}}{2}$; its maximum value is achieved at (i) $\pi_t = \frac{\overline{\pi}}{2}$ if $\frac{\overline{\pi}}{2} \in [\pi_t'', \overline{\pi}]$, and at (ii) $\pi_t = \pi_t''$ if $\frac{\overline{\pi}}{2} < \pi_t''$. In each case, such π_t is the optimal spot price. In the latter case, there are two subcases: (ii.a) if $\underline{\pi} \geq \pi_t'$, the optimal spot price π_t^* is $\underline{\pi}$; (ii.b) if $\underline{\pi} < \pi_t'$, $\pi_t^* = \pi_t'$. In the case (i), the condition $\frac{\overline{\pi}}{2} \in [\pi_t'', \overline{\pi}]$ is equivalent to the condition $\frac{\overline{\pi}}{2} \geq \pi_t''$, which requires both $\frac{\overline{\pi}}{2} \geq \pi_t'$ and $\frac{\overline{\pi}}{2} \geq \underline{\pi}$ by (16); due to $\rho = \underline{\pi}/\overline{\pi}$, this condition is equivalent to the condition $\rho \leq \min\{0.5, 1 - D/2\}$. In the case (ii), the condition is equivalent to $\rho > \min\{0.5, 1 - D/2\}$. The condition $\underline{\pi} \geq \pi_t'$ in the case (ii.a) is equivalent to $D \leq 1$; thus, the conditions to make $\pi_t^* = \underline{\pi}$ are $\rho > \min\{0.5, 1 - D/2\}$ and $D \leq 1$, which are further equivalent to $\rho > \frac{1}{2}$ and $D \leq 1$. Similarly, in the case (ii.b), the conditions to make $\pi_t^* = \pi_t'$ are $\rho > 1 - D/2$ and $D > 1$. Finally, substituting the optimal spot price π_t^* in each case into (18), we could get the maximum revenue $\mathcal{G}^{(i)}(t)$ in (19). □

Finally, we can quantify the revenue improvement α_t and the conclusion below follows directly from (13) and Proposition 5.1.

Corollary 5.2. *The revenue improvement brought by the spot market is as follows:*

$$\alpha_t = \begin{cases} \frac{1}{4} \cdot \frac{1}{1-\rho} \cdot D \cdot I & \text{if } \rho \le \min\{0.5, 1 - D/2\}, \text{ where } \pi_t^* = \overline{\pi}/2 \\ \rho \cdot D \cdot I & \text{if } D \le 1 \text{ and } \rho > 0.5, \text{ where } \pi_t^* = \underline{\pi} \\ (1 - \frac{1-\rho}{D}) \cdot I & \text{if } D > 1 \text{ and } \rho > 1 - D/2, \text{ where } \pi_t^* = \pi_t' \end{cases} \tag{20}$$

where π_t' *is given in (16),* $D = \frac{A_t}{M_t^{(i)}}$, $I = \frac{M_t^{(i)}}{\overline{M}_t/b}$, *and* $\rho = \underline{\pi}/\overline{\pi} \in (0, 1)$.

Now, we explain the physical meaning of Corollary 5.2. Recall that $M_t^{(i)}$ is the number of spot instances available to serve the bids at t, and they are idle instances in the on-demand market; $M_t^{(i)}$ determines the capacity of accepting bids. A_t is the total number of bids at t. $\overline{M}_t$ is the number of instances executing on-demand jobs at t. The ratio $D = A_t/M_t^{(i)}$ can be viewed as *the saturation degree* of spot market at t, e.g., when it is larger than 1, the spot market is fully saturated with bids and not all bids could be accepted with the capacity constraint; when the ratio is zero, there are no bids at t. The ratio ρ is *the value density* of user's bids. If ρ is small, the price difference of users' bids is large. Let us consider a scenario where the maximum bid price of users $\overline{\pi}$ is given and a fixed number of bids with the highest prices are accepted: if ρ is small, the lowest price of the accepted bids would be small; as a result, the spot price at t is also small, as well as the revenue that the CSP gains from the bids. Similarly, we can have opposite conclusions for the case of a large ρ. In the long run, the mean of $\overline{M}_t/b$ approximates the mean of $\overline{M}_t^{(i)}$, and the ratio $I = \frac{M_t^{(i)}}{\overline{M}_t/b}$ could be roughly viewed as *the vacancy-to-utilization ratio* of on-demand market, representing the percentage of servers in idle states at slot t. If the vacancy-to-utilization ratio is large and the CSP does not offer spot service, only a small part of on-demand instances are effectively utilized by on-demand jobs and most of them are in idle states at t; for example, if the ratio is 7, it implies that 87.5% of the instances will be in idle states.

By (20) and (13), the vacancy-to-utilization ratio of on-demand market I, the saturation degree of spot market D, and the value density ρ together determine the revenue improvement α_t after complementing the on-demand market with a spot market. We have by (20) that, the larger the value density ρ, the larger the revenue improvement. For example, in the case that ρ is large and the saturation degree of spot market is low (i.e., $D \le 1$ and $\rho > 0.5$), the best strategy is accepting all bids $\mathcal{A}_t$ at t, achieving the maximum α_t. In the following, we consider another setting where the value density is small (i.e., $\rho \le 0.5$). Suppose that the value density ρ is 0.2. The vacancy-to-utilization ratio of on-demand market is fixed and is mainly determined by the QoS guarantee offered to the arriving on-demand jobs; since the jobs require a quick response from the CSP, its value is usually small, and we set I to 7. Then, we have

$$\alpha_t = \begin{cases} 2.1875 \cdot D = 2.1875 \cdot \left(A_t/M_t^{(i)}\right) & \text{if } D \le 1.6, \\ 7 \cdot \left(1 - 0.8 \cdot \frac{1}{D}\right) = 7 \cdot \left(1 - 0.8 \cdot \left(M_t^{(i)}/A_t\right)\right) & \text{if } D > 1.6. \end{cases} \tag{21}$$

Here, if the saturation degree of spot market is large (i.e., $D = A_t/M_t^{(i)} > 1.6$), we have that after complementing the on-demand market with a spot market, the revenue improvement α_t is no smaller than 3.5, representing at least 3.5-fold increase in the CSP's revenue. If the saturation degree is small (i.e., $A_t/M_t^{(i)} \le 1.6$), the revenue improvement α_t is 2.1875 times the saturation degree of spot market.

Finally, we observe that our derivation and observation above can be in principle generalised by relaxing assumption (ii) and letting $H(\pi_t)$ be a general probability distribution for the bid prices at every slot t over the support $[\underline{\pi}, \overline{\pi}]$; then, the Equation (15) becomes $N_t^{(i)} = A_t \cdot (1 - H(\pi_t))$ and the revenue of spot market in (18) becomes $G^{(i)}(t) = A_t \cdot (1 - H(\pi_t)) \cdot \pi_t / K$; the maximum revenue might be achieved when the spot price π_t is such that the differential of $G^{(i)}(t)$ equals zero. Under any distribution, one may observe that the revenue improvement α_t may mainly depend on the vacancy-to-utilization ratio of on-demand market I, the saturation degree of spot market D, and the value density ρ. For example, given the amount of servers, a larger I means that more servers are idle for accepting bids; given the amount of spot instances and the distribution of bid prices, a larger D means that more bids are available and the CSP can choose to accept the bids with higher prices; in both these cases, a higher revenue improvement may be achieved.

6 PERFORMANCE EVALUATION

In this section we provide numerical validation for the proposed framework.

6.1 Experimental Setting

Time is divided into consecutive slots and each slot contains $k = 5$ minutes. On-demand instances are charged on an hourly basis and an hour contains $L = 12$ slots. We set b to 6, i.e., all servers are divided into 6 groups (also called processing units); for all $i \in [1, 6]$, the i-th group has m_i servers and its value will be given in Section 6.2.1. For all $i \in [1, 6]$, the i-th group is used to process the on-demand and spot jobs that arrive at slots $t = 6 \cdot h + i$ where $h = 0, 1, 2, \cdots$. An exception occurs in Section 6.2 where we takes different values for b to show its effect on the idleness of on-demand market.

6.1.1 Components of On-demand and Spot Services. We have explained in Section 4.3 the basic framework; the final framework divides all servers into 6 groups. For all $i \in [1, 6]$, at the beginning of slot $t = 6 \cdot h + i$ where $h = 0, 1, 2, \cdots$, the job pricing, acceptance, and assignment of the i-th group are similar to the ones of the basic framework, and there are four main components:

1. **Dispatching on-demand jobs.** We use the PTC policy to dispatch jobs to servers, as described in Section 3.2. On-demand users can be latency-critical and user-facing services have to meet strict tail-latency requirements at the 99th percentile of the distribution [8]; in other words, for every 100 jobs, there is at most one job that will miss its deadline. For the i-th processing unit, there are m_i servers where m_i is the minimum number of servers needed to guarantee the latency requirement and its value is given in Section 6.2.1; if the number of servers is larger than m_i, more servers will be idle in the on-demand market and a higher revenue from the spot market may be achieved.
2. **Accepting spot jobs.** As explained in Section 5.2, we apply Algorithm 1 to the i-th group for determining which bids are accepted at slot t.
3. **Assigning spot jobs to servers.** As explained in Section 5.2, we apply the procedure proposed in Section 3.4 to the i-th group for assigning spot jobs at t.
4. **Optimally pricing spot instances.** We use the algorithm SpotiPrice$\left(M_t^{(i)}, \mathcal{A}_t, f_t^{(i)}, K, \beta/b\right)$, described by Algorithm 2, to determine the spot price π_t at t. The time spent on loading or migrating VMIs is set to 3 minutes, i.e., $k' = 3$; here $\beta = \frac{k'}{k} = 0.6$. Finally, we use (9) to determine the revenue from the spot market at t.

6.1.2 Performance Metrics. Recall that α_t is defined in (13) and it is the ratio of the revenue of spot market to the revenue of on-demand market at a slot t. The *average revenue improvement* per slot is the average value of all α_t where $t = 1, 2, 3, \cdots$ and it is denoted by α_e. In our experiments, the

main performance metric is α_e and it represents how much the CSP's revenue is increased by after complementing the on-demand market with a spot market.

Furthermore, for all $i \in [1, 6]$, the i-th processing unit is used to process the jobs arriving at slot $t = 6 \cdot h + i$ where $h = 0, 1, 2, \cdots$. We assume that a super-slot contains 6 slots (i.e., 30 minutes). From the i-th slot on, the server state of the i-th unit changes every super-slot; the h'-th super-slot corresponds to the period of $[t, t + 5]$ where $h' = 1, 2, \cdots$ and $t = 6 \cdot (h' - 1) + i$. In our experiments, we also show the *average server utilization* per super-slot. In the case that the CSP offers both on-demand and spot instances (resp. on-demand instances alone), the utilization at a specific super-slot h' is defined as the ratio of the number of instances occupied by on-demand and spot jobs (resp. on-demand jobs) to the total number of instances available (i.e., m_i), denoted by $\theta_t^{(i)}$, where $h' = (t - i)/6 + 1$. In both cases, the average server utilization per super-slot is simply defined as the average value of $\theta_1^{(1)}, \theta_2^{(2)}, \cdots, \theta_6^{(6)}, \theta_7^{(1)}, \cdots, \theta_{12}^{(6)}, \theta_{13}^{(1)}, \cdots$, which is denoted by θ when only on-demand instances are offered and by μ when both instances are offered. With the values of θ and μ, we can numerically see the improvement to server utilization after complementing the on-demand market with a spot market.

As analyzed in Section 5.3, the revenue improvement may mainly depend on three factors: (i) the vacancy-to-utilization ratio of on-demand market, (ii) the saturation degree of spot market, and (iii) the distribution of users' bid prices. The first factor is mainly determined by the QoS guarantee offered to on-demand users and it is specified in Section 6.1.1; the related results are given in Section 6.2.1. So, the environments of our experiments will vary in terms of the distribution of users' bid prices, and the arrival rate of bids; the main results will be given in Section 6.3.

6.1.3 Arrival of on-demand jobs. A public CSP such as Amazon EC2 serves many users of different sources; it is representative to use a heavy-tailed distribution to model the job size and a poisson distribution to model the job arrival [6, 43], which has been validated by some measurement study [45]. At every slot $t = 1, 2, 3, \cdots$, the number of job arrivals follows a poisson distribution with a mean λ_o. A job's size is a random variable x that follows a bounded pareto distribution with a scale parameter x_m and a shape parameter α; x ranges in $[x_m, \overline{x}]$. Since on-demand instances are charged on an hourly basis, we further set the sizes of the jobs submitted to the CSP to $12 \cdot \lceil \frac{x}{12} \rceil$. For a job j with size s_j and arrival time a_j, its deadline is $d_j = a_j + s_j - 1$. The on-demand price is normalized as 1. In the whole system, there are 6 processing units and for all $i \in [1, 6]$, the i-th unit can be viewed as a single system used to process the jobs arriving at slot $t = 6 \cdot h + i$ where $h = 0, 1, 2, \cdots$. Thus, each unit will process the jobs with the same statistical feature in terms of the arrival rate and the job size. In the experiments, λ_o, x_m and α are set to 60, 6, and $\frac{7}{6}$ respectively; then, the mean of that pareto distribution is 42 (slots); the lower and upper bounds of the job size are 0.5 and 13 hours. An exception occurs in Sec. 6.2.1 where we take different values for λ_o, α, $\overline{x}$ to show their effect on the idleness of on-demand market.

6.1.4 Arrival and departure of spot jobs. At every slot t, there are some users that newly arrive and bid prices for spot instances; among these users, we assume that their bid prices follow some probability distribution. At every t, the bids of these users vary in terms of the arrival rate, and their value distribution. In our simulations we consider two value distributions. The first is a uniform distribution over $[0.2, 1]$, following [46]. The other is a bounded pareto distribution with a pareto index $\alpha = 2$ and a scale parameter $x_m = 0.3$, following [2]; its lower and upper bounds are 0.3 and 1. Given a pareto distribution with $\alpha = 2$ and $x_m = 0.3$, its mean is $\frac{\alpha \cdot x_m}{\alpha - 1} = 0.6$, and the probability that a random variable x takes on a value larger than x_0 is $(\frac{x_m}{x_0})^\alpha$, e.g., the probability that $x > 0.6$ is 0.25. Thus, with the bounded pareto distribution, the proportion of the bids whose prices are small are larger.

All servers are divided into 6 groups; for all $i \in [1, 6]$, the i-th group serves the bids accepted at $t = 6 \cdot h + i$ where $h = 0, 1, 2, \cdots$. The number $x_t^{(i)}$ of the bids that newly arrive at $t = 6 \cdot h + i$ follows a geometric distribution, similar to [46]. Thus, $x_t^{(i)}$ is a random variable that denotes the number of failures before one success in a series of independent trials, where each trial results in either success or failure and the probability of success is the constant $q_i = 1/\lceil m_i/\phi \rceil$. The mean and variance of $x_t^{(i)}$ are $(1 - q_i)/q_i$ and $(1 - q_i)/q_i^2$; roughly, for all $i \in [1, 6]$, the mean bid arrival rate of the i-th processing unit is $\frac{1}{\phi}$ times its processing capacity, i.e., $\lceil \frac{m_i}{\phi} \rceil - 1$. In our simulations, we consider three types of spot market that are respectively *fully*, *moderately*, and *poorly* saturated with bids; correspondingly, the parameter ϕ is set to 1, 2.5, and 5 and its value determines the number of spot jobs that newly arrive at t.

Once the bid of a user is accepted, the assigned instance will dedicate $b = 6$ slots (i.e., 30 minutes) to it. Among the users whose bids are accepted at $t = 6 \cdot h + i$, each will continue bidding at $t + 6$ at a probability $1 - \varrho$ and stop bidding at a probability ϱ; the value of ϱ is chosen from a uniform distribution over $\{0.1, 0.3, 0.5\}$. The mean of ϱ is 0.3; on average, at the beginning of each $t + 6$, 30% of the users accepted at t will stop bidding while the remaining 70% users will continue biding. We denote by $y_{t'}^{(i)}$ the number of the spot users who are accepted at t and will continue bidding at t' where $t' = t + b = 6 \cdot (h + 1) + i$. Thus, at the i-th group of servers, for all $t' = 6 \cdot h + i$ where $h = 1, 2, \cdots$, the total number of bids available is $x_{t'}^{(i)} + y_{t'}^{(i)}$; specially, for the initial slot i, there are only bids that newly arrive and the total number of bids is $x_i^{(i)}$.

6.2 Idleness in On-demand Market

In this subsection, we show the resource utilization when the CSP only provides on-demand instances. This helps better perceive the advantage of selling the idle states of on-demand instances as spot instances, although such practice has been adopted by Amazon EC2. In particular, we will provide both experimental and theoretical results available.

6.2.1 Empirical Results. We implemented a queuing system to reproduce the utilization of servers. The way of generating and dispatching on-demand jobs and the guaranteed quality of services (QoS) are described in Sec. 6.1.3 and Sec. 6.1.1.

We first look at the idleness of on-demand market when the expected job arrival rate λ_o, the shape parameter α, and the upper bound of job size $\overline{x}$ take different values. In a pareto distribution, the larger the value of α, the larger the expected job size; the latter is $\frac{\alpha \cdot x_m}{\alpha - 1}$ for $\alpha > 1$. The experiments are run over a period of about 120000 slots. While a server serves on-demand jobs, some super-slots are unoccupied; the idleness in a period is defined as the ratio of the amount of the unoccupied super-slots of all servers to the amount of the super-slots of all servers, denoted by ϑ. In the first case, we fix $\overline{x} = 156$ and $\alpha = \frac{7}{6}$; the idleness ϑ is 0.8854, 0.8854, and 0.8856 when λ_o equals 30, 60, and 90 respectively, which coincides with the theoretical result that the server utilization is independent of the job arrival rate given the QoS requirement [24]. In the second case, we fix $\lambda_o = 60$ and $\overline{x} = 156$, and the idleness ϑ is 0.8865, 0.8854, and 0.8769 when α equals $\frac{21}{20}$, $\frac{7}{6}$, and $\frac{7}{3}$. In the third case, we fix $\lambda_o = 60$ and $\alpha = \frac{7}{6}$, and the idleness ϑ is 0.8854, 0.8877, and 0.8893 when $\overline{x}$ equals 156, 468, and 1404. The results of the latter two cases imply that the larger the job sizes, the higher the idleness of on-demand market. Under the different conditions above, we can observe that ϑ varies in a very small range of $[0.8769, 0.8893]$; the values of λ_o, α, and $\overline{x}$ have slight effect on the idleness. Complementarily, the experimental results here are also consistent with a measurement study in [21] where some instances of Amazon EC2 are launched and run for one week and the observed server utilization is in the 3% to 17% range.

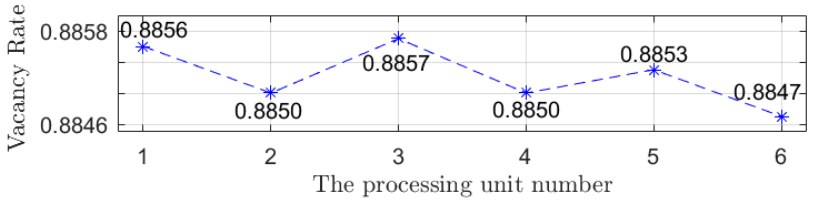

Fig. 9. The vacancy rate of the i-th processing unit when where $b = 6$ and $1 \leq i \leq b$.

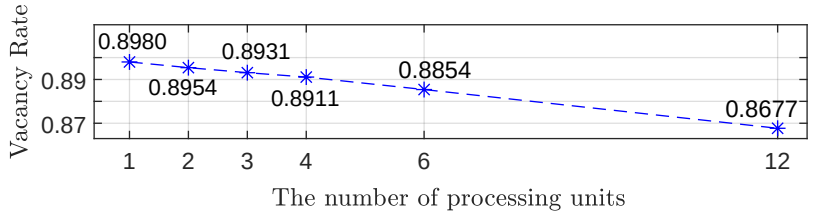

Fig. 10. The vacancy rate of all processing units when the number b of processing units is 1, 2, 3, 4, 6, and 12 respectively.

For all $i \in [1, 6]$, recall the definition of $\theta_t^{(i)}$ in Section 6.1.2. Let $\hat{\theta}_t^{(i)} = 1 - \theta_t^{(i)}$ and it denotes the percentage of servers idle and wasted at a super-slot of the i-th processing unit if the CSP does not sell them as spot instances. The average vacancy rate $\hat{\theta}^{(i)}$ of the i-th unit is defined as the average value of all $\hat{\theta}_t^{(i)}$ where $t = 6 \cdot h + i$ and $h = 0, 1, 2, \cdots$. Now, we fix $\lambda_o = 60$, $\alpha = \frac{7}{6}$, and $\overline{x} = 156$ where the idleness is moderate; the values of $\hat{\theta}^{(1)}, \hat{\theta}^{(2)}, \cdots, \hat{\theta}^{(6)}$ are illustrated in Fig. 9. For example, at the first unit, the percentage of servers idle at every super-slot is 88.56% on average.

In the extended framework of Sec. 5, in order to have Feature 5.1 and Feature 5.2, we divide servers into multiple groups that alternatingly serve the on-demand jobs arriving at different slots; however, in the basic framework of Sec. 3 and Sec. 4, servers are not divided: whenever an on-demand job arrives at any slot, one server will be chosen from all servers to serve it. Now, we show the effect of server division on the server utilization. Each experiment has the same job/workload input, guarantees the same QoS described in Sec. 6.1.1, and is taken respectively with and without server division. We will see that more servers are needed in the non-division scenario; thus, after division, the server utilization is improved and under the extended framework the servers are more effectively utilized by on-demand jobs. In particular, for the first case above, the minimum number of servers needed in the non-division scenario (resp. in the division scenario) is 6823, 13594, and 20421 (resp. 6105, 12198, and 18284); the corresponding utilization is improved by 11.76%, 11.44%, and 11.69%. For the second case, the minimum number of servers needed in the non-division scenario (resp. in the division scenario) is 14589, 13594, and 8876 (resp. 13156, 12198, and 7450); the corresponding utilization is improved by 10.89%, 11.44%, and 19.14%. For the third case, the minimum number of servers needed in the non-division scenario (resp. in the division scenario) is 13594, 15970, and 17837 (resp. 12198, 14452, and 16450); the corresponding utilization is improved by 11.44%, 10.50%, and 8.432%.

Finally, we show the effect of the number b of processing units on the server utilization. Recall the meaning of θ in Section 6.1.2; then, $\hat{\theta} = 1 - \theta$ denotes the average percentage of servers idle at every super-slot of all units, also referred to as the vacancy rate. Recall in Section 5.2 that b needs to be a factor of L. We thus set b to 1, 2, 3, 4, 6, and 12 respectively. Now, we fix $\lambda_o = 60$, $\alpha = \frac{7}{6}$, and

$\overline{x} = 156$, and the vacancy rate $\hat{\theta}$ is illustrated in Fig. 10. We can observe that $\hat{\theta}$ decreases slightly from 0.8980 to 0.8677 as b increases from 1 to 12. On the other hand, we have by Feature 5.2 that the value of b determines the guaranteed duration in which spot jobs can utilize servers once their bids are accepted. Thus, the choice of b depends much on the system designer's experience and view of the quality of spot service.

6.2.2 Theoretical Results. Results from discrete-time queuing theory can be used to help us perceive the relation between the mean waiting time of on-demand jobs and the utilization of servers; The standard definition for a job's waiting time is the queuing time from its arrival to the moment that it gets assigned. In particular, existing literature considers the case where the job arrival at a server follows a geometric distribution and the job size follows a general distribution. In the context of this paper, we can use the round-robin policy in Section 3.2 to uniformly dispatch the arriving on-demand jobs to servers; then, at every server there is a single queue and the mean waiting time of all on-demand jobs will be its counterpart at a server [45]. We denote by σ the job size's standard deviation and by s the mean job size. At a server, the mean waiting time w satisfies the following relation [26]:

$$w = \frac{\lambda \cdot (\sigma^2 + s^2) - \rho}{2 \cdot (1 - \rho)} \tag{22}$$

where one job arrives at a slot with probability λ and the probability that no jobs arrive is $1 - \lambda$ where $\lambda \in [0, 1]$; ρ is the mean utilization or load of a server where $\rho = \lambda \cdot s$. By (22), we also have

$$\frac{1}{\rho} = 1 + \frac{\sigma^2/s + s - 1}{2 \cdot w}. \tag{23}$$

In the following, we illustrate the sensitivity of the server utilization ρ to the waiting time w: the requirement of a small waiting time leads to a low utilization in the on-demand market. In this section, all servers are divided 6 groups and each group can be viewed as a single system where jobs arrive and are dispatched once every super-slot (i.e., 6 slots). To apply the relation (23) directly, we use in this subsubsection the super-slot as the basic time unit for the job size. Since on-demand jobs are charged on an hourly basis, their size will be the multiple of 2 (super-slots); thus $s \geq 2$ and we have by (23) that ρ decreases as w decreases. We assume that, the CSP will guarantee that the mean waiting time w is $\frac{1}{6}$ super-slot. When the job size follows a uniform distribution over $\{2, 4, 6\}$, we have that the mean job size w is 4 and its variance σ^2 is small and equals 2; then, to guarantee the QoS, we have by (23) that the server utilization is $\frac{1}{11.5} \approx 0.08696$. In other words, on average, the servers will be in idle state 91.30% of the time. Thus, many servers are in idle states in the on-demand market, which remain to be utilized by spot users.

6.3 Revenue Improvement of Spot Market

In this subsection, we show the main results of performance evaluation, i.e., the revenue improvement α_e as explained in Sec. 6.1.2. We use $\alpha_e^{(1)}$, $\alpha_e^{(2.5)}$, $\alpha_e^{(5)}$ to denote the α_e respectively in the case that the spot market is fully, moderately and poorly saturated with bids as explained in Sec. 6.1.4. The main experimental results are given in the top subfigure of Fig. 11. For example, in the case that the users' bid prices follow a uniform distribution, if the spot market is saturated with many bids (i.e., the fully-saturated case), there is a at least 4.5-fold increase in the CSP's revenue; if the spot market is less saturated (i.e., the poorly-saturated case), the CSP's revenue can still be increased by more than 1.5-fold.

For the uniform distribution case, the average value of all spot prices is 0.7756, 0.6624, and 0.6061 respectively in the fully, moderately, and poorly saturated spot market. The spot price is the minimum price of all accepted bids. With more bids available, the CSP can choose to accept the bids

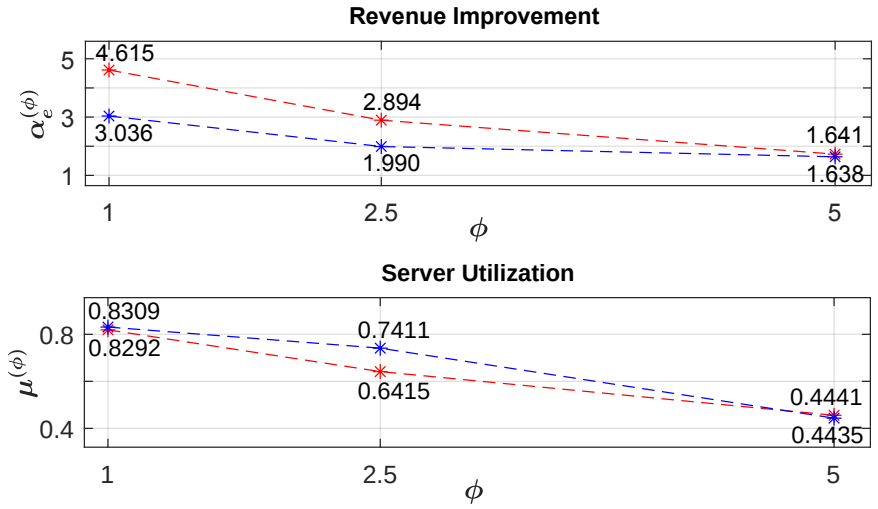

Fig. 11. The revenue improvement $\alpha_e^{(\phi)}$ and server utilization $\mu^{(\phi)}$ when the spot market is fully, moderately, and poorly saturated with bids, corresponding to the cases that $\phi = 1$, 2.5, and 5 respectively: the red (resp. blue) stars are for the case that the bid prices follow a uniform distribution over $[0.2, 1]$ (resp. a bounded pareto distribution over $[0.3, 1]$ with a pareto index of 2); we note that, when the CSP only offers on-demand instances, the server utilization θ is 0.1146.

with higher prices; hence, in a more saturated spot market, the average spot price is also higher. The spot prices of the first unit at slot $t = 222001 + 6 \cdot h' \in [222001, 222300]$ is illustrated in Fig. 12 where $h' = 0, 1, \cdots, 49$; we can observe that, if the spot market is saturated to a higher degree, the change of spot prices over time is also larger. Taking all the 6 processing units into account, the average number of spot jobs accepted per super-slot is 1444, 1065, and 666 respectively in the fully, moderately, and poorly saturated spot market; correspondingly, the average number of spot jobs newly arriving and accepted is 506.1, 335.1, and 201.2. Let us consider the poorly and fully saturated markets respectively. Due to the offer of spot instances to users, in the former case, the average utilization of instances is improved from 0.1146 to 0.4441, while the CSP's revenue is improved by 164.1%; in the latter case, the average utilization of instances is improved from 0.1146 to 0.8292, while the CSP's revenue is improved by 461.5%. As far as the first unit is concerned, the number of spot instances available and the number of accepted bids at slot $t = 222001 + 6 \cdot h' \in [222001, 222300]$ are illustrated in Fig. 13 where $h' = 0, 1, \cdots, 49$.

Finally, we summarize the average server utilization per super-slot before and after offering spot instances to users. Recall the definition of θ and μ in Sec. 6.1.2; we use $\mu^{(1)}$, $\mu^{(2.5)}$, and $\mu^{(5)}$ to denote the server utilization μ when the spot market is fully, moderately, and poorly saturated with bids. The related results are given in the bottom subfigure of Fig. 11.

6.4 Comparison with a Dynamic Reserve Price Algorithm

The framework for sharing the time of servers among on-demand and spot services – in order to optimally pricing spot instances – has four main components as described in Section 6.1.1. The 4-th component is actually the one operating the spot pricing mechanism: consistent with the claim of Amazon EC2 [2], it should rely on the relation of demand and supply, i.e., the bids and the number of spot instances available. We aim at comparing our scheme with the reference one proposed by the authors of [2]; they actually claim that Amazon EC2 may in practice set its spot prices

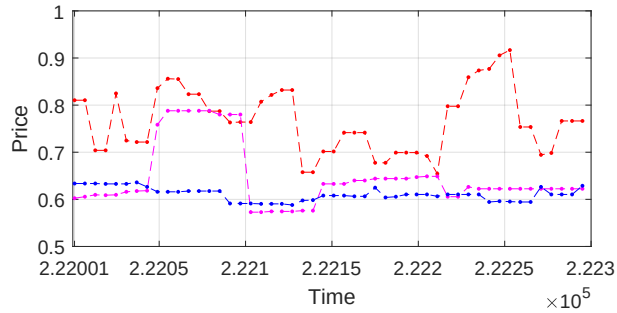

Fig. 12. The spot prices over [222001, 222300]: the red, magenta and blue points correspond to the fully, moderately and poorly saturated case respectively.

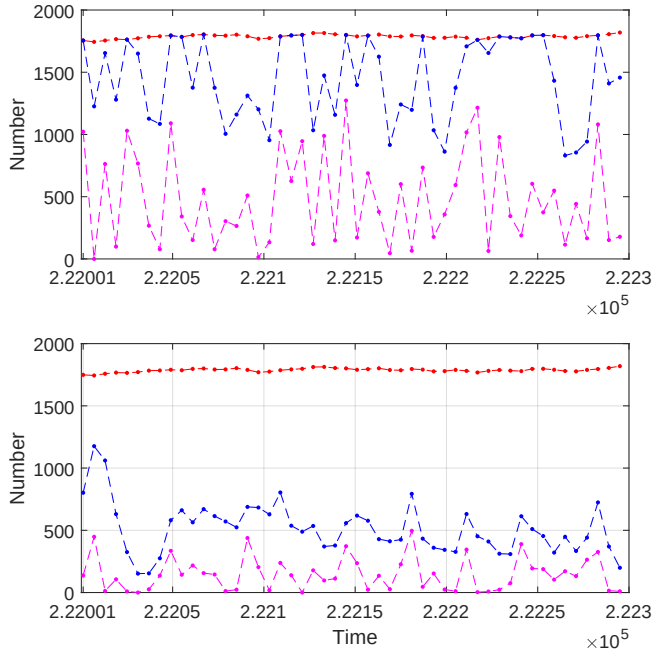

Fig. 13. The supply and demand relation in the slot interval [222001, 222300]: the red points denote the number of spot instances, the blue points denote the total number of bids accepted, and the magenta points denote the number of bids newly arriving and accepted; the top and bottom subfigures correspond to the fully and poorly saturated cases respectively.

artificially by a dynamic reserve price (DRP) algorithm [2]. In this subsection, we thus replace the pricing scheme in the 4th component with the DRP algorithm in [2], and show the performance of our framework when the scheme in [2] is applied. The DRP algorithm draws spot prices from a fixed range $[F, C]$ where F and C are the lower and upper bounds of spot prices. In particular, it is initialized with a reserve price of $P_0 = F$ and a price change of $\Delta_0 = 0.1 \cdot (F - C)$. At each processing unit, the spot price is updated every 6 slots and the l-th spot price P_l is recursively defined as follows where $l = 1, 2, \cdots$:

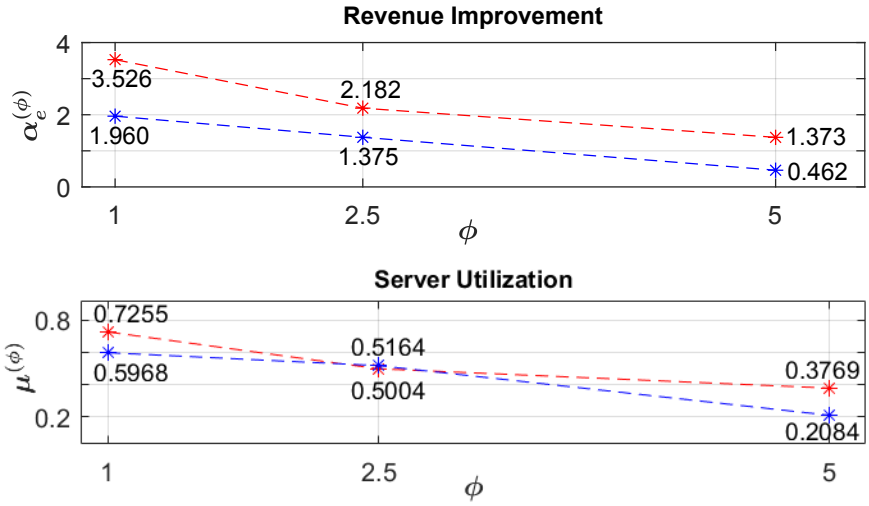

Fig. 14. With the pricing scheme in [2], the revenue improvement $\alpha_e^{(\phi)}$ and server utilization $\mu^{(\phi)}$ when the spot market is fully, moderately, and poorly saturated with bids, corresponding to the cases that $\phi = 1$, 2.5, and 5 respectively: the red (resp. blue) stars are for the case that the bid prices follow a uniform distribution (resp. a bounded pareto distribution); we note that, when the CSP only offers on-demand instances, the server utilization is 0.1146.

$$P_l = P_{l-1} + \Delta_l, \text{ and } \Delta_l = -0.7 \cdot \Delta_{l-1} + \epsilon(\sigma')$$

where $\epsilon(\sigma')$ is white noise with a standard deviation $\sigma' = 0.39 \cdot (C - F)$; here, Δ_l may be generated multiple times until the resulting P_l is within $[F, C]$ and does not equal P_{l-1}, i.e., $P_l \in [F, C] - \{P_{l-1}\}$. In our experiments of Section 6.3, if the users' bid prices follow the uniform distribution, the minimum and maximum spot prices are $(\pi_{min}, \pi_{max}) = (0.5024, 0.9496)$, $(0.5384, 0.8744)$ and $(0.5224, 0.7712)$ respectively in the fully, moderately, and poorly saturated case; correspondingly, if the bid prices follow the pareto distribution, (π_{min}, π_{max}) is $(0.3000, 0.8110)$, $(0.3000, 0.6110)$ and $(0.5048, 0.7272)$ respectively. In the experiments, we set $(F, C) = (\pi_{min}, \pi_{max})$ in each case. The related results for revenue improvement are given in the top subfigure of Fig. 14, and the server utilization after complementing the on-demand market with a spot market is given in the bottom subfigure of Fig. 14; the meaning of related symbols has been introduced in Section 6.3.

Overall, we can see from the top subfigures of Fig. 11 and Fig. 14 that, the proposed algorithm of this paper can achieve higher revenue improvement than the DRP algorithm in [2]; here, the revenue from the on-demand market depends on the utilization of on-demand market that equals 0.1146. As claimed in [2], when the spot market is saturated with less bids, the use of the DRP scheme can artificially create a false impression of the changes of demand and supply and mask times of low demand and price inactivity, thus possibly driving up the CSP's stock. This is confirmed in our experimental results: as illustrated in Fig. 12, when the spot market is saturated with many bids (e.g., the fully-saturated case), the spot prices vary more dramatically over time; however, in the poorly-saturated case, the spot prices vary slightly and even keep constant in a relatively long period. So, in the poorly-saturated case, it may be necessary to set spot price artificially.

6.5 Comparison with Another Framework for On-demand and Spot Services

We first introduce the framework of Dierks *et al.* in [10, 11] where on-demand and spot markets are modeled as two separate queues Q_1 and Q_2. For Q_1, the number of servers m_o is the minimum severs needed to guarantee that the expected waiting time of jobs is very small; the price of utilizing a sever is p per unit of time. The second Q_2 is a priority queue that has m_S servers: every job continuously bids to utilize the servers of spot market until it gets enough execution time; jobs with higher bid prices have higher priorities to utilize servers. There are n job classes whose expected arrival rates are $\lambda = (\lambda_1, \cdots, \lambda_n)$. The job size is drawn from a probability distribution with expectation $\frac{1}{\mu}$. For every job of class i, a waiting cost c is drawn from a distribution $F_i(c)$ on $[0, v_i']$. For a job of spot market, its execution on a server may be preempted when there are unfinished jobs of higher bid prices; each preemption brings a cost $c \cdot \tau$ to it. Let $\sigma = (\zeta, \eta)$ and σ denotes the strategy of a job: when $\zeta = \mathcal{O}$, $\mathcal{S}$, or $\mathcal{B}$, it means that this job will choose the on-demand, spot, or neither market; η is the bid price if $\zeta = \mathcal{S}$. In a Bayesian Nash incentive compatible spot market framework, all participants ideally have the knowledge such as λ, μ and $F_1(c), \cdots, F_n(c)$. Further, each job of class i can derive the optimal strategy σ to maximize its expected payoff; here, if a job chooses spot market, its bid price will be its waiting cost c and its payment can also be derived. With the payments of jobs, we can get the revenue of both markets.

The framework of this paper elaborates on the current service model in Amazon EC2. Compared with [11], the Amazon EC2 model has its advantages in terms of revenue generation and quality of service. It allows selling the idle state of on-demand market on the spot market to get additional revenue [2, 9]. A complication of the model in [11] lies in that the completion time of a spot job depends on the arrival rate of the jobs of higher bid prices that is usually uncertain in reality. This incurs the volatility of jobs' completion times and even delay-tolerant users can become reluctant to accept such service. As discussed in Sec. 2.1, using the model of this paper, delay-tolerant jobs can first bid to utilize spot instances in some period and then turn to stable on-demand instances, leading to that they are finished at expectable times. The numerical comparison of revenue should be taken under the same input condition. In [11], once a spot job submits the bid, it cannot cancel its bid until it gets a specific amount of execution time; its payment relies on its completion time and waiting cost. In our framework, a spot user can stop bidding at any latter slot after it begins bidding; the payment at a slot is the minimum price of all accepted bids, with no connection to the job's waiting cost and completion time. The CSP's revenue is the price times the processed workload. To enable comparison, we let the average spot prices in both frameworks be the same. Specifically, we let the on-demand price be 1; the generated revenue is 0.5 when a spot instance dedicates an hour to a job. As before, we use the PTC policy to assign jobs to servers.

In [11], we set $n = 1$ and the waiting costs of jobs are uniformly distributed on $\{0.3, 0.7\}$. The job arrivals follow a poisson distribution, with expectations λ_o' in on-demand market and λ_s' in spot market. Fixing $\lambda_o' = 30$, we consider three cases with $\lambda_s' = 30$, 60, and 120 respectively. The sizes of jobs are set as described in Sec. 6.1.3. Jobs of spot market have to be finished within bounded periods. We let jobs with $c = 0.7$ have a waiting time ≤ 48 slots (i.e., 4 hours) and let jobs with $c = 0.3$ have a waiting time ≤ 132 slots (i.e., 11 hours); the value of m_S is the minimum amount of servers needed to guarantee the QoS, i.e., the spot market is fully saturated with as many jobs as possible; the QoS is guaranteed at the 99th percentile as described in Sec. 6.1.1. The total number of servers is $m = m_o + m_s$. We compute the average numbers of servers utilized per slot in on-demand and spot markets, denoted by $\overline{m}_o$ and $\overline{m}_s$; the average unit revenue of both markets is $(\overline{m}_o + 0.5 \cdot \overline{m}_s)/12$. For comparison, we also compute the average unit revenue achieved under our framework in the case that there are m servers; here, the spot market is also fully saturated with all idle servers of on-demand market being utilized by spot users. We denote by $\hat{\alpha}_e$ the ratio of the average unit

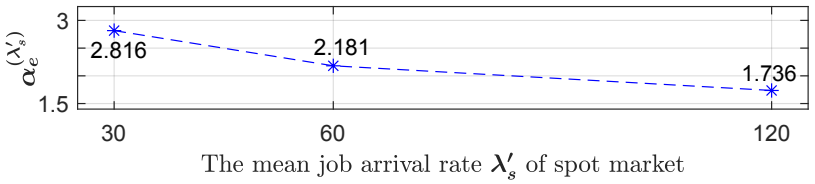

Fig. 15. Revenue improvement compared with the on-demand and spot model in [11].

revenue of our framework to its counterpart with the framework of [11]; let $\alpha_e = \hat{\alpha}_e - 1$ denote the revenue improvement when comparing our framework with the one in [11]. We use $\alpha_e^{(30)}$, $\alpha_e^{(60)}$, $\alpha_e^{(120)}$ to denote the α_e respectively in the case that λ_s' is 30, 60, and 120, and the experimental results are given in Fig. 15.

7 CONCLUSION

In a system where on-demand and spot users coexist, on-demand users arrive randomly and have high priority to access servers, while spot users bid prices to utilize the time periods unoccupied by on-demand users. A key feature to make such services accessible is that, on-demand users can get served within a short time upon arrivals, while a spot user can stably utilize a server (without the interference of on-demand users) for a sufficient amount of time once bidding successfully. In this paper, we propose a framework that has such a feature for sharing the time of servers among on-demand and spot users. Under such a framework, specific schemes are proposed to accept and assign the requests of on-demand and spot users to servers and to optimally price spot instances. The framework itself is designed under assumptions which are met in real environments. With a few further mild assumptions, an analysis of the proposed framework is also taken to understand which parameters drive its performance. Extensive simulations show a significant improvement to the revenue as well as the server utilization once an on-demand market is complemented with a spot market. In the case where less bids are available, the revenue improvement is showed to be indeed smaller, but still significant compared with a bare on-demand market.

REFERENCES

[1] Vineet Abhishek, Ian A Kash, and Peter Key. 2012. Fixed and Market Pricing for Cloud Services. In *the 7th Workshop on the Economics of Networks, Systems, and Computation (NetEcon'12)*. IEEE, 157–162.

[2] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. 2011. Deconstructing Amazon EC2 Spot Instance Pricing. In *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom'11)*. IEEE, 304–311.

[3] Amazon.com, Inc. 2018. Amazon EC2 pricing. (2018). https://aws.amazon.com/ec2/purchasing-options/ (accessed on Nov. 28, 2018).

[4] Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph (Seffi) Naor, and Jonathan Yaniv. 2015. Truthful Online Scheduling with Commitments. In *Proceedings of the 16th ACM Conference on Economics and Computation (EC'15)*. ACM, 715–732.

[5] Omar Besbes and Assaf Zeevi. 2009. Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms. *Operations Research* 57, 6 (2009), 1407–1420.

[6] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. 2011. Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. ACM, 229–238.

[7] Andrew Chung, Jun Woo Park, and Gregory R. Ganger. 2018. Stratus: Cost-aware Container Scheduling in the Public Cloud. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'18)*. ACM, 121–134.

[8] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating*

Systems (ASPLOS'14). ACM, 127–144.

[9] Nikhil R. Devanur. 2017. A Report on the Workshop on the Economics of Cloud Computing. *ACM SIGecom Exchanges* 15, 2 (2017), 25–29.

[10] Ludwig Dierks and Sven Seuken. 2016. Cloud Pricing: The Spot Market Strikes Back. In *the Workshop on the Economics of Cloud Computing (in conjunction with ACM EC'16)*. ACM, 1–28.

[11] Ludwig Dierks and Sven Seuken. 2019. Cloud Pricing: The Spot Market Strikes Back. In *Proceedings of the 20th ACM Conference on Economics and Computation (EC'19)*. ACM, 1–35.

[12] Haoming Fu, Zongpeng Li, Chuan Wu, and Xiaowen Chu. 2014. Core-Selecting Auctions for Dynamically Allocating Heterogeneous VMs in Cloud Computing. In *Proceedings of the 2014 IEEE International Conference on Cloud Computing (CLOUD'14)*. IEEE Computer Society, 152–159.

[13] Gartner, Inc. 2018. Gartner Says Worldwide IaaS Public Cloud Services Market Grew 29.5 Percent in 2017. (2018). https://www.gartner.com/newsroom/id/3884500 (accessed on Nov. 28, 2018).

[14] Tim Hellemans, Tejas Bodas, and Benny Van Houdt. 2019. Performance Analysis of Workload Dependent Load Balancing Policies. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 2, Article 35 (2019), 35 pages.

[15] Gabriel Iuhasz, Pooyan Jamshidi, Weikun Wang, and Giuliano Casale. 2017. Load Balancing for Multi-cloud. In *Model-Driven Development and Operation of Multi-Cloud Applications*. Springer, 53–58.

[16] Navendu Jain, Ishai Menache, Joseph (Seffi) Naor, and Jonathan Yaniv. 2015. Near-Optimal Scheduling Mechanisms for Deadline-Sensitive Jobs in Large Computing Clusters. *ACM Transactions on Parallel Computing* 2, 1, Article 3 (2015), 29 pages.

[17] Brendan Jennings and Rolf Stadler. 2015. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management* 23, 3 (2015), 567–619.

[18] Konstantinos Karanasos, Sriram Rao, Carlo Curino, Chris Douglas, Kishore Chaliparambil, Giovanni Matteo Fumarola, Solom Heddaya, Raghu Ramakrishnan, and Sarvesh Sakalanaga. 2015. Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters. In *Proceedings of the 2015 Usenix Annual Technical Conference (USENIX ATC '15)*. USENIX Association, 485–497.

[19] Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. 2018. A Survey on Spot Pricing in Cloud Computing. *Journal of Network and Systems Management* 26, 4 (2018), 809–856.

[20] Zheng Li, He Zhang, Liam O'Brien, Shu Jiang, You Zhou, Maria Kihl, and Rajiv Ranjan. 2016. Spot Pricing in the Cloud Ecosystem: A Comparative Investigation. *Journal of Systems and Software* 114 (2016), 1–19.

[21] Huan Liu. 2011. A Measurement Study of Server Utilization in Public Clouds. In *Proceedings of the IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC'11)*. IEEE Computer Society, 435–442.

[22] Ming Mao and Marty Humphrey. 2012. A Performance Study on the VM Startup Time in the Cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing (CLOUD'12)*. IEEE Computer Society, 423–430.

[23] Ishai Menache, Ohad Shamir, and Navendu Jain. 2014. On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud. In *the 11th International Conference on Autonomic Computing (ICAC'14)*. USENIX, 177–187.

[24] Michael Mitzenmacher. 2001. The Power of Two Choices in Randomized Load Balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104.

[25] Debankur Mukherjee, Souvik Dhara, Sem C. Borst, and Johan S.H. van Leeuwaarden. 2017. Optimal Service Elasticity in Large-Scale Distributed Systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 1, Article 25 (2017), 28 pages.

[26] NPTEL. 2012. Discrete Time Queues: Geo/G/1 Queue - Late Arrival Model. (2012). https://nptel.ac.in/courses/117103017/45 (accessed on Jan. 17, 2019).

[27] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. 2013. Sparrow: Distributed, Low Latency Scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*. ACM, 69–84.

[28] Jeff Rasley, Konstantinos Karanasos, Srikanth Kandula, Rodrigo Fonseca, Milan Vojnovic, and Sriram Rao. 2016. Efficient Queue Management for Cluster Scheduling. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys'16)*. ACM, 15.

[29] Kaveh Razavi, Liviu Mihai Razorea, and Thilo Kielmann. 2013. Reducing VM Startup Time and Storage Costs by VM Image Content Consolidation. In *European Conference on Parallel Processing*. Springer, 75–84.

[30] Weijie Shi, Chuan Wu, and Zongpeng Li. 2014. RSMOA: A revenue and social welfare maximizing online auction for dynamic cloud resource provisioning. In *Proceedings of the 2014 IEEE/ACM 22nd International Symposium of Quality of Service (IWQoS'14)*. IEEE, 41–50.

[31] Weijie Shi, Chuan Wu, and Zongpeng Li. 2017. An Online Auction Mechanism for Dynamic Virtual Cluster Provisioning in Geo-Distributed Clouds. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (March 2017), 677–688.

[32] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Francis C.M. Lau. 2014. An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing. In *Proceedings of the 2014 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '14)*. ACM, 71–83.

[33] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Francis C. M. Lau. 2016. An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing. *IEEE/ACM Transactions on Networking* 24, 4 (2016), 2060–2073.

[34] Peijian Wang, Yong Qi, Dou Hui, Lei Rao, and Xue Liu. 2013. Present or Future: Optimal Pricing for Spot Instances. In *Proceedings of the IEEE 33rd International Conference on Distributed Computing Systems (ICDCS'13)*. IEEE Computer Society, 410–419.

[35] Weikun Wang and Giuliano Casale. 2014. Evaluating Weighted Round Robin Load Balancing for Cloud Web Services. In *Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'14)*. IEEE, 393–400.

[36] Wei Wang, Ben Liang, and Baochun Li. 2013. Revenue maximization with dynamic auctions in IaaS cloud markets. In *Proceedings of the 2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS'13)*. IEEE, 1–6.

[37] Xiaohu Wu and Francesco De Pellegrini. 2017. On the Benefits of QoS-Differentiated Posted Pricing in Cloud Computing: An Analytical Model. (2017). arXiv:1709.08909

[38] Xiaohu Wu and Patrick Loiseau. 2015. Algorithms for scheduling deadline-sensitive malleable tasks. In *Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton'15)*. IEEE, 530–537.

[39] Xiaohu Wu, Patrick Loiseau, and Esa Hyytiä. 2017. Towards Designing Cost-Optimal Policies to Utilize IaaS Clouds with Online Learning. In *Proceedings of the 2017 International Conference on Cloud and Autonomic Computing (ICCAC'17)*. IEEE, 160–171.

[40] Hong Zhang, Hongbo Jiang, Bo Li, Fangming Liu, Athanasios V. Vasilakos, and Jiangchuan Liu. 2016. A Framework for Truthful Online Auctions in Cloud Computing with Heterogeneous User Demands. *IEEE Trans. Comput.* 65, 3 (2016), 805–818.

[41] Linquan Zhang, Zongpeng Li, and Chuan Wu. 2014. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *Proceedings of the IEEE Interational Conference on Computer Communications (INFOCOM'14)*. IEEE, 433–441.

[42] Xiaoxi Zhang, Zhiyi Huang, Chuan Wu, Zongpeng Li, and Francis C. M. Lau. 2017. Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs. *IEEE/ACM Transactions on Networking* 25, 2 (2017), 1034–1047.

[43] Xiaoxi Zhang, Chuan Wu, Zhiyi Huang, and Zongpeng Li. 2018. Occupation-Oblivious Pricing of Cloud Jobs via Online Learning. In *Proceedings of the IEEE Interational Conference on Computer Communications (INFOCOM'18)*. IEEE, 2456–2464.

[44] Xiaoxi Zhang, Chuan Wu, Zongpeng Li, and Francis CM Lau. 2019. A Truthful $(1-\epsilon)$-Optimal Mechanism for On-demand Cloud Resource Provisioning. *IEEE Transactions on Cloud Computing* (2019).

[45] Liang Zheng, Carlee Joe-Wong, Christopher G. Brinton, Chee Wei Tan, Sangtae Ha, and Mung Chiang. 2016. On the Viability of a Cloud Virtual Service Provider. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science (SIGMETRICS'16)*. ACM, 235–248.

[46] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, and Xinyu Wang. 2015. How to Bid the Cloud?. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*. ACM, 71–84.

[47] Ruiting Zhou, Zongpeng Li, and Chuan Wu. 2018. An Online Emergency Demand Response Mechanism for Cloud Computing. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3, 1, Article 5 (2018), 25 pages.

[48] Ruiting Zhou, Zongpeng Li, Chuan Wu, and Zhiyi Huang. 2016. An Efficient Cloud Market Mechanism for Computing Jobs With Soft Deadlines. *IEEE/ACM Transactions on Networking* 25, 2 (2016), 793–805.