# Burstable Instances for Clouds: Performance Modeling, Equilibrium Analysis, and Revenue Maximization

Yuxuan Jiang, *Member, IEEE,* Mohammad Shahrad, *Member, IEEE,* David Wentzlaff, *Member, IEEE,* Danny H.K. Tsang, *Fellow, IEEE,* and Carlee Joe-Wong, *Member, IEEE*

***Abstract*—Leading cloud providers recently introduced a new instance type named *burstable* instances to better match the time-varying workloads of tenants and further reduce their costs. In the research community, however, little has been done to understand burstable instances from a theoretical perspective. This paper presents the first unified framework to model, analyze, and optimize the operation of burstable instances. Specifically, we model the resource provisioning of burstable instances, identify key performance metrics, and derive the analytical performance given the resource provisioning decisions. We then characterize the equilibrium behind tenants' responses to the prices offered for different burstable instance service classes, taking into account the impact of tenants' actions on the performance achieved by each service class. In addition, we investigate how a cloud provider can leverage knowledge of this equilibrium to find the prices that maximize its total revenue. Finally, we validate our framework on real traces and demonstrate its usage to price burstable offerings in a public cloud.**

***Index Terms*—cloud, burstable instances, equilibrium, revenue maximization**

## I. Introduction

To reduce costs for cloud tenants, today's Infrastructure-as-a-Service (IaaS) providers offer various pricing schemes, such as on-demand pricing, spot pricing, and reserved pricing [2]. Under these pricing schemes, however, tenants always obtain virtual machines (VMs) provisioned with static amounts of resources, for example, one virtual CPU (vCPU) and 2 GB memory. On the other hand, empirical studies [3], [4], [5], [6] have reported that workloads executed on VMs in public clouds are usually time-varying. Therefore, given the static amount of resources provisioned for VMs, tenants have to book VM configurations that can satisfy their peak workload demands. This peak-demand subscription strategy leads to low actual utilization of the resources allocated to VMs. Take CPU resource utilization as an example. The utilization is lower than 35% on average according to a Google cluster trace study [4], and lower than 20% for 60% of the VMs according to a Microsoft Azure trace study [5]. These observations imply that tenants' costs can be further reduced by time-varying resource provisioning. In other words, VMs receive a high volume of resources for a short period of time in exchange for fewer resources most of the time. A new class of VMs, named *burstable* instances, has thus been introduced by a number of cloud providers, such as the *t2* and *t3* instances of Amazon EC2 [7], *B-series* instances of Microsoft Azure [8], and *f1-micro* and *g1-small* instances of the Google Cloud Engine [9]. In this paper, we approach burstable instances from a theoretical perspective, and present the first unified framework to model, analyze, and optimize their operation. We show that cloud providers can use this framework to understand the performance of burstable instances and set the corresponding prices to optimize their total revenue.

A preliminary version appeared at the IEEE International Conference on Computer Communications (INFOCOM), Paris, France, 2019 [1]. This work was supported in part by the NSF under Grants No. CNS-1751075 and CCF-1453112, Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7846, FA8650-18-2-7852, and FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA), the NSF, or the U.S. Government.

Y. Jiang and D.H.K. Tsang are with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong (e-mails: {yjiangad@connect, eetsang@ece}.ust.hk).

M. Shahrad is with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada, V6T 1Z4 (e-mail: mshahrad@ece.ubc.ca). He was with the Department of Electrical Engineering, Princeton University, NJ 08544, USA during this work.

D. Wentzlaff is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA (e-mail: wentzlaf@princeton.edu).

C. Joe-Wong is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA (e-mail: cjoewong@andrew.cmu.edu).

TABLE I: Samples of Microsoft Azure burstable instances [8].

| Type | CPU credits earned per hour | Maximum CPU credits buffered | Resource volume (vCPUs) | |
|---|---|---|---|---|
| | | | Maximum | Mean |
| B1ls | 3 | 72 | 1 | 0.05 |
| B1s | 6 | 144 | | 0.1 |
| B1ms | 12 | 288 | | 0.2 |

### A. Background on Burstable Instances

We list a few sample burstable instance configurations in Table I. A burstable instance has a resource budget quantified by CPU credits. Each CPU credit provides 100% of the full capacity of a vCPU for a time slot's duration (e.g., 1 minute in Microsoft Azure [8]). CPU credits can be used in fractions, such as spending 0.1 CPU credits for 10% of a vCPU. The credits are earned at a constant rate per time slot for an instance, with a limit on the maximum number of credits that can be buffered. The maximum resource volume is the maximum amount of resources that an instance can receive in a time slot, which is one vCPU for all instances in Table I. On the other hand, the rate of credit earning determines the average resource volume (sometimes also referred to as "baseline") for an instance. For example, a *B1ls* instance in Table I receives 0.05 CPU credits per time slot (i.e., 1 minute), enabling it to request 5% of a vCPU on average over time.

Burstable instances are suitable for services that demand relatively small amounts of resources most of the time, while occasionally requiring large amounts of resources. For example, VMs operating as hot standbys [10] are usually idle with low CPU utilizations. When a failover occurs, they demand high resources to take over the jobs, but only for a short while until the normal services are recovered. Applications with periodic workloads, such as periodically updating machine learning models [11], are also suitable for burstable instances.

Compared to traditional static resource provisioning methods, burstable instances can benefit both tenants and cloud providers. Tenants no longer need to pay for their peak resource demands all the time, so their costs are potentially reduced with fewer resources purchased. Cloud providers can also benefit in terms of over-commitment.[1] Though widely employed, over-commitment traditionally suffers from the difficulty of understanding VMs' CPU utilization patterns, which providers do not control [4]. Therefore, providers have to co-locate VMs in a relatively conservative manner to offer a guaranteed quality-of-service (QoS) level, i.e., the chance that a VM can successfully receive its requested resources [12]. The CPU utilization of burstable instances, however, is regulated by the CPU credit mechanism, making the utilization patterns more predictable for providers. Providers may then be able to co-locate more burstable instances on a server while still offering a guaranteed QoS level. Moreover, by jointly optimizing the offered QoS and the prices charged to the tenants, providers can maximize their total revenues. In this paper, we provide a framework for them to do so.

## B. Our Contributions

Although cloud computing with static resource provisioning has been extensively studied, burstable instances are still an emerging research topic with many unanswered questions. Consider a cloud provider that offers different types of burstable instances for multiple tenants. Hereinafter, we refer to tenants as users, and to instance types as service classes defined by the configuration parameters shown in Table I. In this paper, we aim to understand three fundamental questions on burstable instances and use them to help cloud providers *(i)* estimate the performance of burstable instances, and *(ii)* increase their total revenue for operating this service.

**How can we define and analytically evaluate the performance of burstable instances?** The QoS that a burstable instance receives is determined by the amount of resources that a VM is allocated compared to how many it requests, i.e., how well a user's resource needs can be fulfilled. Note that the QoS depends on whether the user's requests are allowed by the CPU credit mechanism, as well as how the cloud provider multiplexes its (over-committed) resources. Therefore, analytically formulating the QoS representation is non-trivial as it requires us to mathematically translate the CPU credit mechanism to CPU utilization patterns and integrate the result with the resource multiplexing scheme. To this end, in Section II, we first formally define the QoS metric. We model the dynamics of CPU credits as a token bucket regulation mechanism [13]. Meanwhile, we model two resource multiplexing schemes for burstable instance services, *random selection* and *proportional allocation*, and finally derive analytical QoS representations for both of these multiplexing schemes.

**From an individual user's perspective, which service class should (s)he select to maximize his/her reward?** We proceed to look at an IaaS cloud that offers burstable instances with multiple service classes, each configured by CPU credit parameters and a resource capacity. We refer to these parameters as service class configurations hereinafter. A service class charges a price to each user who subscribes to it. Note that a rational user always favors a service class that offers higher QoS with lower payment. Therefore, the user will select the service class where his/her reward, which can be regarded as his/her valuation of the received QoS minus the payment, is maximized. In Section III, we analytically derive users' service class selections at the Nash equilibrium.

**From a cloud provider's perspective, how should it price the service classes to maximize its total revenue?** The equilibrium derived above characterizes users' responses (i.e., service class selections) to the prices offered by service classes, accounting for individual users' heterogeneous QoS valuations. Note that a cloud provider's total revenue depends on both the number of users subscribed to each service class and the prices that the users should pay for their subscriptions. Given the service class configurations, a cloud provider can thus set prices leveraging prior knowledge of the equilibrium on users' corresponding subscription decisions to maximize its total revenue at equilibrium.[2] In Section IV, we formulate a mixed-integer non-linear program to obtain such optimal prices for the provider. While this problem can be solved by general-purpose methods for mixed-integer programs [14], we also propose an algorithm to compute an approximate solution in a more efficient manner.

Our answers to the three questions above constitute a framework to model, analyze, and optimize burstable instance services in IaaS clouds. In Section V, we numerically validate our framework using real-world traces [5] and show that it can drastically improve the cloud provider's total revenue compared to heuristic pricing methods.

The remainder of the paper is organized as follows. In Sections II, III, and IV, we answer the three aforementioned questions sequentially, and simultaneously develop our frame-

[1]Over-commitment in clouds means the resources allocated to the VMs on a server can exceed the server's actual capacity, if the VMs are expected not to fully utilize their reserved resources simultaneously [4]. Therefore, VMs may not always receive the full resources that they demand.

[2]Our revenue maximization problem does not capture the temporal evolution in the number of users. However, the number of users does not change much over time according to the state-of-the-art traces [5]. To apply our proposed methods to real-world public clouds, we can use the user profiles at the peak for pricing. Although conservative, the derived prices and total revenue are still shown to be reasonably good (see Section V-C for details).

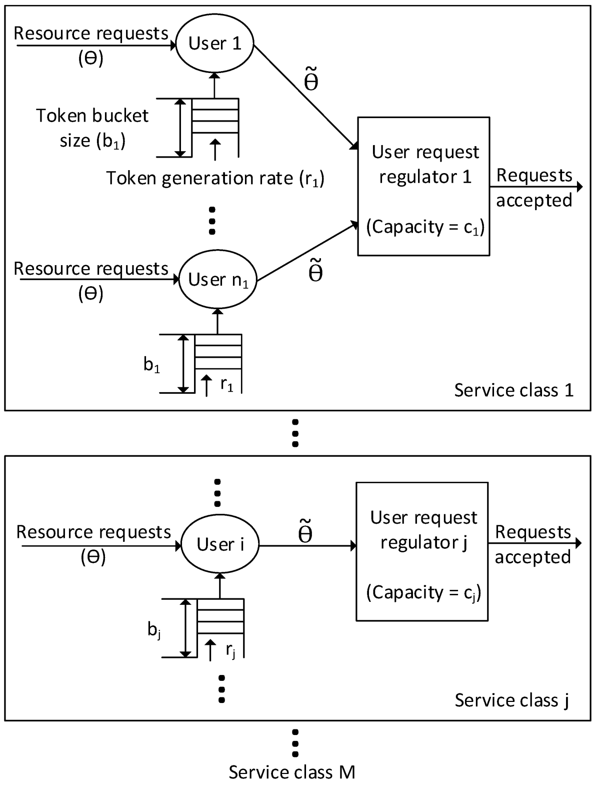


Fig. 1: The system model of $M$ service classes in a cloud. A user corresponds to a VM in the cloud and maintains a token bucket. Users subscribing to the same service class share a regulator that multiplexes the burstable resources. A resource request either gets a token from the token bucket and proceeds to the regulator, or is discarded because there are no longer any available tokens in the token bucket. The regulator guarantees that the burstable resources allocated to VMs do not exceed the capacity of a service class.

work. We numerically validate our framework and study one of its use cases in Section V. Related works are surveyed in Section VI. We conclude the paper in Section VII. Due to the limited space, we will present the derivations of all lemmas, corollaries, and propositions of Section II and Section III in Appendix B and Appendix C, respectively.

## II. Performance Modeling

In this section, we first introduce the system model of an IaaS cloud offering burstable instances with multiple service classes. Then we formally define our QoS metric and analytically derive the QoS that a service class can offer, in terms of the number of users subscribing to it and the service class configurations specified by the cloud provider.

### A. System Model

We study a slotted time system with $N$ users and $M$ service classes. The system model is shown in Figure 1. In the cloud, each service class provides a certain level of QoS. A user is associated with a burstable instance running in the cloud, and can subscribe to a service class. Suppose service class $j \in \mathcal{M} = \{1, 2, ..., M\}$ has $n_j$ subscribed users. According to an empirical study on burstable instances [13], the life-cycle of CPU credits for each user is essentially the same as that of the tokens in the token bucket model [15], [16], [17]. We thus employ a dedicated token bucket for each user, as shown in Figure 1. In our model, we convert CPU credits to a more fine-grained unit named tokens. A token stands for the smallest resource unit that a user can request in a time slot. In the beginning of a time slot, $r_j$ tokens are first generated to the token bucket of each user subscribing to service class $j$. As an example of the CPU-credit-to-token conversion, suppose a token stands for 1% of the full capacity of a vCPU, and the duration of a time slot is one minute. A *B1ls* instance shown in Table I receives 0.05 CPU credits per time slot, equivalent to five tokens. The maximum number of buffered tokens for a user subscribing to service class $j$ is $b_j$. Moreover, an instance is guaranteed to receive $a^{grntd}$ token units[3] of resources to maintain its underlying essentials, such as the operating system. Therefore, even if an instance's requests are all rejected in a time slot (for example, due to too many simultaneous requests from peer instances), the instance will not halt due to receiving insufficient resources. If we set $a^{grntd} = 0$, our model falls back to the basic burstable instance design presented in Table I. Since guaranteed resources are always available and are the same for each service class, they do not affect users' QoS, and thus $a^{grntd}$ is not included in our following performance model. Apart from guaranteed resources, in each time slot, an instance can make requests for more resources. Service class $j$ reserves $c_j$ token units of resources for the requests from all its subscribers. If the total amount of requested resources from users exceeds the resource capacity of a service class, the limited resource capacity should be allocated to users following a *resource multiplexing scheme*, which will be discussed later in this sub-section. To sum up, the configuration of a service class $j$ can be characterized by three parameters, $b_j$, $c_j$, and $r_j$, which are set by the cloud provider. Without loss of generality, we assume that $b_j$, $c_j$, and $r_j$ are positive integers in this paper.

We continue to illustrate how users' requests are made and processed in each time slot. Each request is for one token unit of resources. For example, if the user wants 5% of a vCPU for a time slot, with one token standing for 1% of a vCPU, a batch of five requests will be made. A user's requests are processed as soon as new tokens for this time slot are accumulated. As Figure 1 shows, each request checks if there is an available token in the token bucket; if so, the request proceeds to the user request regulator and the corresponding token is deducted from the user's token bucket. Otherwise, the request will be discarded. Therefore, when the number of available tokens in the token bucket is smaller than the number of incoming requests in a time slot, only a partial number of requests, which equals the number of tokens currently available in the token bucket, can proceed to the regulator, and the remaining requests are discarded. The user may make new requests in the coming time slots if (s)he still demands burstable resources.

[3]We refer to a token unit as a resource unit representing the amount of resources corresponding to one token.

On the other hand, when the number of tokens in the token bucket is no smaller than the number of incoming requests, all of these requests can proceed to the regulator. This model extends our earlier work [1], where all requests are simply discarded without getting any tokens if the number of available tokens in the token bucket is smaller than the number of incoming requests.

At service class $j$'s user request regulator, all requests will be approved if the total number of received requests does not exceed the resource capacity, $c_j$. Otherwise, the regulator will allocate the limited resources to users following some *resource multiplexing scheme*. We can consider two schemes, *random selection* and *proportional allocation*. In *random selection*, the regulator repeatedly chooses a user uniformly at random, and admits his/her requests until the capacity $c_j$ is used up. Extending our earlier work [1], we also consider *proportional allocation*, where each user receives an equal proportion (e.g., 90%) of his/her requested resources, so that the total amount of resources allocated to users sums to the resource capacity, $c_j$. Hence, a user may eventually receive a fractional amount of resources. This paper will derive analytical results for both of the multiplexing schemes and provide insights into their numerical results so that a cloud provider can have a better idea of which scheme to choose for its real operation.

### *B. Quantifying Users' QoS*

In this paper, we are interested in the analytical form of a user's received QoS by subscribing to a service class. For simplicity, we assume that users are homogeneous in the statistical patterns of their requests. The probability that a user has at least one request in a time slot is denoted by $\delta$. The number of requests that a user makes in a time slot, given that (s)he has requests to make, is a random variable $\theta \in [1, \theta_{max}] \cap \mathbb{Z}^+$.[4] For example, if a token stands for 1% of a vCPU, with the maximum resource volume as one vCPU and guaranteed resources as 2% of a vCPU (i.e., $a^{grntd} = 2$) for each instance, an instance can request up to 98% of a vCPU as its burstable resources (i.e., $\theta_{max} = 98$). To simplify our mathematical model, we suppose that $b_j > 2\theta_{max}$, as is typically the case in practice.[5] Denote the probability that $\theta$ takes the value $x$ by $P(\theta = x)$. The distribution of $\theta$ can be estimated from historical data of CPU utilization for a particular application. We assume that $P(\theta = x) > 0$, $x \in [1, \theta_{max}] \cap \mathbb{Z}^+$. This assumption will be later confirmed by real-world traces [5] in Section V.

By making $\theta$ requests, a user $i \in \mathcal{N} = \{1, 2, ..., N\}$ subscribing to service class $j$ finally receives $\phi_j$ token units of resources, where $\phi_j$ is a random variable (that depends on $\theta$). Note that $\phi_j$ depends not only on $b_j$, $c_j$, and $r_j$, the service class configurations, but also on $n_j$, the number of peer users that are concurrently sharing the resources in service class $j$ with this user. Let $\tilde{\theta} \in [1, \tilde{\theta}_{max}] \cap \mathbb{Z}^+$ be the number of requests that can traverse the token bucket and reach the regulator for a given user in a time slot, given that the user makes requests in this time slot. Note that $\tilde{\theta}$ is a random variable with the same range as that of $\theta$ (i.e., $\tilde{\theta}_{max} = \theta_{max}$). Meanwhile, $\tilde{\theta}$ is always non-negative, because the token bucket has at least $r_j \geq 1$ available tokens (the ones accumulated in the same time slot) to accommodate potential requests. Given that a user makes $\theta$ requests, there must be at least $\min\{\theta, r_j\}$ requests that can traverse the token bucket. We denote the probability that $\tilde{\theta}$ takes the value $y$ by $P\left(\tilde{\theta} = y\right)$.

The QoS that user $i$ subscribing to service class $j$ receives, denoted by $q_j$, is defined as the probability that the user can finally receive $\alpha$ $(0 \leq \alpha \leq 1)$ of his/her requested resources in a time slot given that (s)he makes requests in this time slot, where $\alpha$ is a cloud-provider-specified parameter. In other words, provided that a user makes $\theta = x$ requests in a given time slot, $q_j$ is the probability that this user can finally receive no fewer than $\alpha x$ token units of resources. Practically, $\alpha$ can be a fractional number that is close to 1, for example, 0.9. In this case, our performance metric characterizes the probability that a user receives 90% of his/her requested resources. Our QoS metric guarantees the tail probability on the fraction of a user's resource requests that are ultimately fulfilled. This tail probability guarantee strategy has been widely adopted in the cloud computing literature [18], [19].

Similar to existing studies on IaaS clouds [6], [20], our QoS metric quantifies the resource availability on the infrastructure level instead of modeling the application-level performance (e.g., job completion time). A CPU credit (a.k.a. a token) can be interpreted as an *opportunity* that a user can spend to obtain resources. Our QoS metric represents the probability that the CPU credits can finally turn into the allocated resources. To understand the infrastructure-level QoS, users who deploy their applications on the cloud-provided VMs may need to translate the infrastructure-level QoS to the application-level performance. Although such a translation is out of the scope of our paper, which targets an IaaS cloud, we still provide a few translation examples in Appendix A.

We make comments from a qualitative perspective on how the service class configurations $b_j$, $c_j$, and $r_j$ will affect the QoS for a given number of users in service class $j$. Generally, increasing $b_j$ and $r_j$ at the token bucket side leads to a higher chance of the users' requests passing the token bucket, as Figure 1 shows. If the regulator's capacity $c_j$ is underutilized, the QoS of users will improve due to the increases in $b_j$ and $r_j$. However, if the capacity is already overutilized with a multiplexing scheme in place, to improve the QoS, we should also increase $c_j$ to adapt it to the increased number of requests that reach the regulator.

We continue to derive the analytical QoS. Assume the system is stationary. We can formulate $q_j$ by enumerating the probabilities that a user makes $x$ resource requests, with $y$ requests successfully traversing the token bucket, and finally

[4]For some applications, the number of requests made in a time slot may be temporally correlated to that made in previous time slots. However, an IaaS cloud has neither the knowledge of what applications are running on the VMs nor the control of these applications. Therefore, our user request model does not consider such temporal correlation. We will show in Section V that our model is still accurate for realistic user request patterns from state-of-the-art public cloud traces [5], which may not be i.i.d. across time.

[5]As typical values, suppose one token stands for 1% of a vCPU and the maximum resource volume for an instance is one vCPU. Let $a^{grntd} = 0$, so $\theta_{max}$ will go up to 100. Practically, the token bucket size is the total number of tokens that can be buffered within 24 hours (see Table I). Therefore, even if the token generation rate is $r_j = 1$, with the duration of a time slot as one minute, the token bucket size is $b_j = 1440$, much greater than $\theta_{max}$.

$$q_j = \sum_{x=1}^{\theta_{max}} P\left(\phi_j \geq \alpha x | \theta = x\right) P\left(\theta = x\right) = \sum_{x=1}^{\theta_{max}} \sum_{y=\lceil \alpha x \rceil}^{x} P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right) P\left(\tilde{\theta} = y | \theta = x\right) P\left(\theta = x\right) \quad (1)$$

receives at least $\alpha x$ token units of resources, as equation (1) at the top of this page. In equation (1), $P\left(\phi_j \geq \alpha x | \theta = x\right)$ is the probability that a user initially makes $x$ requests and finally receives at least $\alpha x$ token units of resources, $P\left(\tilde{\theta} = y | \theta = x\right)$ is the probability that exactly $y$ requests successfully traverse the token bucket for a user, given that (s)he initially makes $x$ requests, and $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$ is the probability that a user finally receives no fewer than $\alpha x$ token units of resources given that (s)he makes $x$ requests and $y$ requests successfully traverse the token bucket. Consider a user who initially makes $x$ requests, with $y$ requests successfully traversing the token bucket. In order for this user to receive no fewer than $\alpha x$ token units of resources in the end, a necessary condition is that $y \geq \alpha x$. Since $y$ is an integer, in the inner summation with regard to $y$ in equation (1), we only consider the situations when $\lceil \alpha x \rceil \leq y \leq x$.

It can be observed from equation (1) that the value of $q_j$ depends on the following two factors: *(i)* How many requests can get the corresponding tokens and traverse the token bucket, characterized by $P\left(\tilde{\theta} = y | \theta = x\right)$; *(ii)* Whether enough requests that have already traversed the token bucket can be admitted by the regulator after multiplexing with other peer users' requests, characterized by $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$. In the rest of this section, we will model the token bucket mechanism and the resource multiplexing scheme at the regulator side to get the expression of $P\left(\tilde{\theta} = y | \theta = x\right)$ and $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$, respectively.

*1) Modeling the token bucket mechanism:* We model the dynamics of the token bucket as a Markov chain, with the state defined as the number of tokens in the token bucket in a time slot, after $r_j$ tokens are generated, but before the potential requests are made and processed. In this case, the token bucket has at least $r_j$ tokens, and thus we have $b_j - r_j + 1$ states. Let state $d = r_j, r_j + 1, ..., b_j$ be the $(d - r_j + 1)$th state of the Markov chain with $d$ tokens in the bucket. The state transition probabilities of the Markov chain are given by Proposition 1. Note that although this proposition is based on our prior assumption of $b_j > 2\theta_{max}$, our derived results can be easily generalized to the cases where $\theta_{max} \leq b_j \leq 2\theta_{max}$ using the same methodology. (For each service class, $b_j \geq \theta_{max}$ should always hold to meet the users' requests.)

**Proposition 1.** *The transition probability $P_{d\to h}$ from state $d$ to state $h$ for the Markov chain is as follows:*

*(i) When $r_j \leq d \leq \theta_{max} - 1$, we have*

$$P_{d\to h} = \begin{cases} \delta \cdot \sum_{k=d}^{\theta_{max}} P(\theta = k) & h = r_j, \\ \delta \cdot P\left(\theta = d + r_j - h\right) & r_j + 1 \leq h \\ & \leq d + r_j - 1, \\ 1 - \delta & h = d + r_j, \\ 0 & otherwise. \end{cases}$$

*(ii) When $\theta_{max} \leq d \leq b_j - r_j$, we have*

$$P_{d\to h} = \begin{cases} \delta \cdot P\left(\theta = d + r_j - h\right) & d + r_j - \theta_{max} \leq h \\ & \leq d + r_j - 1, \\ 1 - \delta & h = d + r_j, \\ 0 & otherwise. \end{cases}$$

*(iii) When $b_j - r_j + 1 \leq d \leq b_j$, we have*

$$P_{d\to h} = \begin{cases} \delta \cdot P\left(\theta = d + r_j - h\right) & d + r_j - \theta_{max} \\ & \leq h \leq b_j - 1, \\ \delta \cdot \sum_{k=1}^{d+r_j-b_j} P\left(\theta = k\right) + (1-\delta) & h = b_j, \\ 0 & otherwise. \end{cases}$$

The Markov chain is positive recurrent and aperiodic, so it is ergodic [21]. Denote the steady-state probability of state $d$ by $\pi_j^d$. The physical meaning of $\pi_j^d$ is the steady-state probability that there are $d$ tokens available in the token bucket waiting for potential requests to be processed. We can obtain $\pi_j^d$ by solving the balance equation.

With the Markov chain model above, we are ready to derive $P\left(\tilde{\theta} = y | \theta = x\right)$. Note that we always have no fewer than $r_j$ tokens in the token bucket when requests arrive. For this reason, when $1 \leq x \leq r_j$, all of the requests can traverse the token bucket, which means

$$P\left(\tilde{\theta} = y | \theta = x\right) = \begin{cases} 1 & y = x, \\ 0 & otherwise. \end{cases} \quad (2)$$

When $r_j < x \leq \tilde{\theta}_{max}$, however, depending on the token availability in the token bucket, either all or a part of the requests can traverse the token bucket, and thus

$$P\left(\tilde{\theta} = y | \theta = x\right) = \begin{cases} \sum_{d=x}^{b_j} \pi_j^d & y = x, \\ \pi_j^y & r_j \leq y < x, \\ 0 & otherwise. \end{cases} \quad (3)$$

With the expressions of $P\left(\tilde{\theta} = y | \theta = x\right)$ derived above, we can further construct the probability mass function $P\left(\tilde{\theta} = y\right)$ of random variable $\tilde{\theta}$ by enumerating all the possible numbers of requests that a user initially makes, namely,

$$P\left(\tilde{\theta} = y\right) = \sum_{x=1}^{\theta_{max}} P\left(\tilde{\theta} = y | \theta = x\right) P\left(\theta = x\right). \quad (4)$$

Substituting equations (2) and (3) into (4), we obtain the full representation of $P\left(\tilde{\theta} = y\right)$ as

$$P\left(\tilde{\theta} = y\right) = \begin{cases} P\left(\theta = y\right) & 1 \leq y < r_j, \\ P\left(\theta = y\right) \left(\sum_{d=y}^{b_j} \pi_j^d\right) + \sum_{x=y+1}^{\theta_{max}} P\left(\theta = x\right) \pi_j^y \\ & r_j \leq y \leq \tilde{\theta}_{max}. \end{cases} \quad (5)$$

$$P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right) = \sum_{k=0}^{n_j-1} \binom{n_j-1}{k} \delta^k (1-\delta)^{n_j-k-1} \left( \frac{1}{k+1} \sum_{h=1}^{k+1} P\left( \sum_{l=1}^{h-1} \tilde{\theta}_l \leq c_j - \alpha x \right) \right) \tag{6}$$

$$P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right) = \sum_{k=0}^{n_j-1} \binom{n_j-1}{k} \delta^k (1-\delta)^{n_j-k-1} P\left( \sum_{l=1}^{k} \tilde{\theta}_l \leq \left( \frac{c_j}{\alpha x} - 1 \right) y \right) \tag{7}$$

The first and second term in equation (5) for $r_j \leq y \leq \tilde{\theta}_{max}$ corresponds to the cases where at least $y$ tokens are available with $y$ requests being made, and where only $y$ tokens are available with more than $y$ requests being made, respectively.

We proceed to derive $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$ in equation (1). To this end, we will respectively model the two multiplexing schemes, *random selection* and *proportional allocation*.

*2) Modeling the regulator's resource multiplexing scheme:* Since we assume that users are homogeneous, it suffices to derive $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$ from the perspective of an individual user, who is referred to as the examined user hereinafter. In what follows, we model *random selection* and *proportional allocation*.

**Random selection.** Equation (6) at the top of this page shows the analytical form of $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$ under *random selection*. First, the amount of resources that the examined user can obtain is directly related to the number of requests that also reach the regulator from other peer users. We denote the number of such peer users by $k$ in equation (6), with the corresponding probability of occurrence as $\binom{n_j-1}{k}\delta^k (1-\delta)^{n_j-k-1}$(recall that $\delta$ is the probability a user makes at least one request), which are the first three terms inside the outer summation of equation (6).

The *random selection* scheme can be equivalently described as the following. The regulator keeps selecting a user uniformly at random to admit his/her requests until all the users' requests are admitted or the residual capacity is used up. In the latter case, the regulator will use its residual capacity to partially satisfy the requests from the last selected user. Following this scheme, given that there are $k+1$ users in total (including $k$ peer users and the examined user) making requests in the service class, the probability that the examined user is the $h$th ($1 \leq h \leq k+1$) user to be selected by the regulator is $1/(k+1)$. Each peer user $l$ ($1 \leq l \leq h-1$) has $\tilde{\theta}_l$ requests reaching the regulator, where $\tilde{\theta}_l$ is a random variable that can be obtained from equation (5). The $h-1$ previously selected users have $\sum_{l=1}^{h-1} \tilde{\theta}_l$ requests reaching the regulator in total. Therefore, the probability that this examined user is admitted with no fewer than $\alpha x$ token units of resources allocated is $P\left(\sum_{l=1}^{h-1} \tilde{\theta}_l \leq c_j - \alpha x\right)$.

**Proportional allocation.** Equation (7) at the top of this page shows the analytical form of $P\left(\phi_j \geq \alpha x | \tilde{\theta} = y, \theta = x\right)$ under *proportional allocation*. Similar to *random selection*, we examine the $\sum_{l=1}^{k} \tilde{\theta}_l$ requests reaching the regulator made by the $k$ peer users. The range of $\sum_{l=1}^{k} \tilde{\theta}_l$ is $[k, k\tilde{\theta}_{max}]$. Recall that the examined user now has $y$ requests reaching the regulator. Here, $y$ should be no smaller than $\alpha x$ to meet the QoS requirement, as the index of the inner summation in equation (1) shows. If $\sum_{l=1}^{k} \tilde{\theta}_l$ is no more than $c_j - y$, the total number of requests received by the regulator does not exceed its capacity. In this case, all the requests will be admitted, and the examined user will get $y$ token units of resources. Otherwise, each user will get his/her share of the total capacity $c_j$ proportional to his/her requested resources. In other words, the amount of resources that the examined user receives is $c_j y/(y + \sum_{l=1}^{k} \tilde{\theta}_l)$. In view of the QoS metric, we need to ensure that

$$\frac{c_j}{y + \sum_{l=1}^{k} \tilde{\theta}_l} y \geq \alpha x,$$

which yields

$$\sum_{l=1}^{k} \tilde{\theta}_l \leq \left( \frac{c_j}{\alpha x} - 1 \right) y.$$

To sum up, given that $k$ users in service class $j$ have requests in a time slot, the probability that the examined user can get no fewer than $\alpha x$ token units of resources is

$$P\left( \sum_{l=1}^{k} \tilde{\theta}_l \leq \left( \frac{c_j}{\alpha x} - 1 \right) y \right),$$

which is the last term inside the outer summation of equation (7).

By sequentially substituting equations (2), (3), and (5) into (6) or (7), and finally into (1), we obtain the analytical form of $q_j$. When referring to equation (1) as the analytical QoS for burstable instances in the rest of this paper, we mean its complete representation after all the above substitutions.

## III. Equilibrium Analysis

Given the service class configuration parameters $b_j$, $c_j$, and $r_j$, in the last section, we have derived the relationship between the QoS $q_j$ that a service class $j$ can deliver with respect to its number of subscribers $n_j$, as shown in equation (1). When operating multiple service classes of burstable instances, the cloud provider assigns a price $p_j$ for each service class $j$ to charge to the corresponding subscribers. In this section, we continue to understand the users' responses, i.e., their preferred selections of service classes, to the prices issued by the cloud provider. Specifically, let each user $i \in \mathcal{N}$ specify a coefficient $u_i$ that represents his/her valuation of the received QoS (e.g., relationship between the received QoS and the resulting application-level performance); the user will therefore harvest a utility of $u_i q_j$ by subscribing to service class $j$. Following prior works in the network economics literature [22], [23], we assume that each user's $u_i$ value lies on a continuum with range $(0, \gamma]$. Let $f(x)$ be the cumulative distribution function

(CDF) of the random variable $u_i$ at $x \in (0, \gamma]$. Denote the reward that user $i$ earns by subscribing to service class $j$ by $w_{i,j}$, which can be calculated by the user's harvested utility minus payment, i.e.,

$$w_{i,j} = u_i q_j - p_j. \tag{8}$$

We focus on deriving the *Nash equilibrium* of users' service class selections. From an individual user $i$'s perspective, at the Nash equilibrium, (s)he should receive more reward from his/her selected service class, denoted by $\eta(i)$, than from other service classes, which can be mathematically written as

$$w_{i,\eta(i)} \geq \max\left\{w_{i,j}, 0\right\}, \forall j \in \mathcal{M}\backslash\{\eta(i)\}. \tag{9}$$

Let $\eta(i) = 0$ if user $i$ decides not to subscribe to any service class. According to equation (9), this can happen if all the service classes deliver negative rewards to this user.

Suppose a Nash equilibrium exists and has been reached. (The existence of a Nash equilibrium will be proved in Proposition 2 later in this section.) Each service class $j$ has $n_j$ subscribers and delivers a QoS of $q_j$ according to equation (1). We next analytically characterize the relationship among $n_j$, $p_j$, $q_j$, $u_i$, and $\eta(i)$, $i \in \mathcal{N}$, $j \in \mathcal{M}$, at the Nash equilibrium.

Our first result finds a sufficient condition that a service class has no subscribers:

**Lemma 1.** *Consider two service classes, $j$ and $k$, where $p_j < p_k$. If $q_j \geq q_k$, then $n_k = 0$ at equilibrium.*

We know from Lemma 1 that for a service class that charges a higher price but offers a lower QoS than another service class at equilibrium, the former service class has essentially no subscribers. The following corollary elaborates a similar idea for two service classes that charge the same price.

**Corollary 1.** *For two service classes, $j$ and $k$, where $p_j = p_k$, if $q_j > q_k$, then $n_k = 0$ at equilibrium.*

As Corollary 1 suggests, if two service classes charge the same price but offer different QoS at equilibrium, the service class that offers a lower QoS has essentially no subscribers. Note that we should prevent a service class from having no subscribers at equilibrium because the cloud provider will derive no profit from it. Therefore, we learn from Lemma 1 and Corollary 1 that the prices of service classes should be properly set so that at the Nash equilibrium *(i)* a service class that charges a higher price should also offer a higher QoS, and *(ii)* the service classes that charge the same price should offer the same QoS. In the rest of this section, we assume that the prices of the service classes are set as aforementioned. Without loss of generality, we index the service classes in non-decreasing order with respect to the QoS that they offer at equilibrium (i.e., $q_j \leq q_k$, $\forall j < k$ and $j, k \in \mathcal{M}$). Since we properly set the prices, we have $p_j \leq p_k$, $\forall j < k$ and $j, k \in \mathcal{M}$. Also, if $p_j = p_k$, $j, k \in \mathcal{M}$, then $q_j = q_k$. Also, we suppose that if two users, $i$ and $k$ with $u_i < u_k$, have decided to subscribe to different service classes with the same offered QoS (and thus the same price) at equilibrium, their subscriptions follow $\eta(i) < \eta(k)$.

From an individual user's perspective, we continue to derive which service class the user will subscribe to. In the next lemma, we qualitatively illustrate the relationship between users' QoS valuations and their service class selections as a necessary condition for a Nash equilibrium.

**Lemma 2.** *Suppose user $i$ selects service class $\eta(i)$. For any user $k$ with a QoS valuation $u_k > u_i$, his/her service class selection $\eta(k)$ satisfies $\eta(k) \geq \eta(i)$.*

We can also derive a sufficient condition when each user has an incentive to subscribe to a service class.

**Corollary 2.** *Each user will have an incentive to subscribe to a service class at equilibrium (i.e., $\forall i \in \mathcal{N}$, $\exists j \in \mathcal{M}$, $w_{i,j} \geq 0$) if $p_1 = 0$.*

Lemma 2 shows that users' service class selections are monotonic with regard to their QoS valuations: users with higher QoS valuations $u_i$ will subscribe to service classes with higher QoS levels (i.e., higher indices) at equilibrium. In other words, as users' $u_i$ values lie on a continuum within the region $(0, \gamma]$, we can partition the region into multiple non-overlapping intervals $(0, v_0)$, $[v_{j-1}, v_j)$, $j \in \mathcal{M} \setminus \{M\}$, and $[v_{M-1}, v_M]$, where $v_M = \gamma$. Users with QoS valuations $u_i \in [v_{j-1}, v_j)$, $j \in \mathcal{M} \setminus \{M\}$ will subscribe to service class $j$, while users with $u_i \in [u_{M-1}, u_M]$ will subscribe to service class $M$. When $p_1 > 0$, users with $u_i \in (0, v_0)$ do not have incentives to subscribe to any service class. On the other hand, if $p_1 = 0$, then $v_0 = 0$ according to Corollary 2, meaning that all users will have incentives to subscribe to service classes. Given this quantitative description, we can fully characterize users' service class selections by determining the boundary points $\{v_j,\ j = 0, 1, ..., M-1\}$ of the intervals, at which a user is indifferent to the choice between the neighboring service classes. We can establish the relationship between the number of subscribers $n_j$ of service class $j$ and the corresponding boundary points $v_{j-1}$ and $v_j$ as

$$n_j = N\left(f\left(v_j\right) - f\left(v_{j-1}\right)\right),\ j \in \mathcal{M}. \tag{10}$$

Note that $f(v_M) = f(\gamma) = 1$. Define $n_0$ as the number of users who have no incentive to join any service class. We have

$$n_0 = Nf(v_0). \tag{11}$$

With the users' service class selections defined above, we can analytically characterize a Nash equilibrium as follows.

**Proposition 2.** *Equation (12) serves as a necessary and sufficient condition that $p_j$, $q_j$, $v_j$, and $n_j$, $j \in \mathcal{M}$, constitute a Nash equilibrium:*

$$p_j = v_0 q_1 + \sum_{k=2}^{j} v_{k-1}\left(q_k - q_{k-1}\right),\ \forall j \in \mathcal{M}. \tag{12}$$

From an individual user's perspective, the Nash equilibrium finds the best trade-off in achieving a high utility $u_i q_j$ with a low payment $p_j$, as defined in equation (9). If user $i$ attaches more importance to the received QoS, (s)he should also have a higher affordability. The user can then take a higher value of $u_i$. At equilibrium, the user will be assigned to a service class that delivers a higher QoS, which correspondingly charges a higher price. Otherwise, the user should take a smaller value of $u_i$, which will potentially lead to a lower QoS delivered and a lower price charged to the user at equilibrium.

From the cloud provider's perspective, the Nash equilibrium characterizes users' corresponding responses (i.e., service class selections) to the prices set by the provider. In the next section, we will continue to study how to take advantage of the knowledge of this equilibrium to set optimal prices.

## IV. Revenue Maximization for The Cloud Provider

The equilibrium derived from Section III provides an opportunity for the cloud provider to maximize its total revenue via optimal pricing. More specifically, with prior knowledge of the relationship between users' service class selections and prices, as given in Proposition 2, the cloud provider can indirectly control the number of users in each service class via setting the corresponding prices. Because the total revenue of the provider is related to both the prices and the actual numbers of users subscribed to the service classes, we can define a revenue maximization problem to find the prices that maximize the provider's total revenue at the Nash equilibrium.

We optimize the provider's total revenue given the service class configurations $b_j$, $c_j$, and $r_j$. Also, we consider $n_0$ and $v_0$, which characterize the provider's preference in accepting users, as pre-specified by the cloud provider. (For example, a provider that wishes to accommodate every user will take $v_0 = 0$, with all users being accepted.) We also let the provider specify another parameter, $\tau$, which indicates the minimum QoS that each service class should offer. Let $\mathbf{p} = [p_1, \ldots, p_M]^T$, $\mathbf{q} = [q_1, q_2, \ldots, q_M]^T$, $\mathbf{n} = [n_1, n_2, \ldots, n_M]^T$, and $\mathbf{v} = [v_1, \ldots v_{M-1}]^T$ be the concatenated vectors of decision variables. With the performance model and user selection equilibrium respectively defined in Sections II and III, we can formulate the revenue maximization problem as

$$\begin{aligned} \underset{\mathbf{p},\mathbf{q},\mathbf{n},\mathbf{v}}{\text{maximize}} \quad & \sum_{j=1}^{M} p_j n_j, && (13)\\ \text{subject to} \quad & \text{constraints (1), (10), and (12)}, \\ & q_j \le q_{j+1},\ \forall j \in \mathcal{M} \setminus \{M\}, && (14)\\ & q_1 \ge \tau, && (15)\\ & n_j \in \mathbb{Z}^+,\ \forall j \in \mathcal{M}. && (16) \end{aligned}$$

In the objective function (13), the provider's total revenue is the summation over the revenue $p_j n_j$ gained by each service class $j$. Together with constraints (1) and (10), constraint (12) defines the relationship among the decision variables at the Nash equilibrium. Since vector $\mathbf{q}$ is sorted in a non-decreasing order at equilibrium, without loss of generality, we configure the service classes as $b_j \le b_{j+1}$, $c_j \le c_{j+1}$, $r_j \le r_{j+1}$, $j \in \mathcal{M} \setminus \{M\}$. Therefore, it is natural to expect that service classes with richer resources will offer higher QoS levels, as indicated in constraint (14). Meanwhile, constraints (14) and (15) jointly guarantee that the QoS $q_j$ offered by each service class $j$ satisfies the minimum requirement $\tau$.

Following similar models from the network economics literature [22], [23], we use users' statistical characteristics (i.e., existing the CDF of $u_i$) for optimal pricing, as shown in constraint (14). The realizations of users' utility parameters $u_i$

**Algorithm 1** Approximation Algorithm for the Revenue Maximization Problem in Section IV.

```
Input: Service class configurations $\{b_j, c_j, r_j,\ \forall j \in \mathcal{M}\}$,
    provider-specified parameters $n_0$ and $v_0$, user profiles $N$,
    $\delta$, $\theta$, QoS metric parameter $\alpha$, and the pre-calculated
    $\{n_j^{upper},\ j \in \mathcal{M}\}$.
Output: p, q, n, and v.
1: Relax constraint (16) as a continuous constraint:
       $n_j \ge 0,\ \forall j \in \mathcal{M}.$          (17)
2: Linearly or quadratically approximate constraint (1), and
   also constraint (10) if it is neither linear nor quadratic.
3: Construct and solve the semidefinite relaxed formulation
   of the revenue maximization problem.
4: Recover feasible solution p, q, n, and v to the original
   revenue maximization problem from the optimal solution
   to the semidefinite relaxed formulation in Step 3.
5: return p, q, n, and v.
```

may not exactly match the distribution $f(\cdot)$, where the actual revenue achieved by the cloud provider may deviate from that derived from our optimization problem. However, this deviation will be negligible when the total number of users, $N$, is large. We can thus interpret the optimal revenue derived by our optimization problem as the "expected" revenue (which we still refer to as the revenue hereinafter for brevity) given users' statistical characteristics. Our optimal solution also guarantees constraint (14) with probability one for realizations of $u_i$.

As a mixed-integer non-linear program, our revenue maximization problem is a hard problem in general, which can incur a high computational complexity to get an optimal solution by existing general-purpose solution algorithms for mixed-integer programs in the literature (e.g., brute-force search) [14]. Therefore, in the rest of this section, we also propose Algorithm 1, an approximation algorithm, to compute an approximate solution for the optimization problem in a more efficient manner. Details of the algorithm are illustrated as follows.

Taking a close look at the problem structure, we find that the optimization problem is almost an inhomogeneous quadratically constrained quadratic program (QCQP)[6] except that we know the exact form of neither the performance model in equation (1) nor the CDF $f(\cdot)$ in equation (10). Therefore, the core notion of our algorithm is to construct an approximate QCQP of the optimization problem, and then apply semidefinite relaxation (SDR) [24] to relax a few constraints towards an efficiently solvable convex optimization problem. Specifically, we first construct an inhomogeneous QCQP of the original optimization problem by relaxing the discrete constraint (16) as a continuous constraint (17) and approximating constraint (1), indicated by step 1 and step 2, respectively, in Algorithm 1. Depending on the actual form of $f(\cdot)$, we also need to approximate constraint (10) if it is neither linear nor quadratic (but not necessarily convex). We

[6]According to the definition in [24], an inhomogeneous QCQP is a QCQP with linear terms in its objective function and/or constraints.

TABLE II: Service class configurations.

| $j$ | $r_j$ | $b_j$ | $c_j$ | Resource volume (vCPUs) | | $a^{grntd}$ |
|---|---|---|---|---|---|---|
| | | | | Maximum | Mean | |
| 1 | 4 | $1,152$ | 100 | 1 | 0.05 | 1% of a vCPU |
| 2 | 6 | $1,728$ | 200 | | 0.07 | |
| 3 | 8 | $2,304$ | 300 | | 0.09 | |
| 4 | 14 | $4,032$ | 400 | | 0.15 | |
| 5 | 19 | $5,472$ | 500 | | 0.2 | |

present Algorithm 1 as a general framework that allows any approximation method for a linear or quadratic approximation. The reason for not allowing higher-order approximations is that the resulting formulation after approximation has to be an inhomogeneous QCQP so that SDR can be applied in the following steps. We will demonstrate in Section V the detailed approximation method that we use in deriving our numerical results. As SDR is a widely-used technique, we do not elaborate the details of constructing a semidefinite relaxed formulation in step 3 here, but refer interested readers to Appendix D-A. The constructed convex optimization problem can be efficiently solved by well-developed algorithms (e.g., interior-point methods [25]). In step 4, we recover a feasible solution to the original revenue maximization problem from the optimal solution to the semidefinite relaxed problem. The detailed algorithm that we use for recovery is presented in Appendix D-B.

## V. Numerical Validation and Case Study

Above, we have defined our theoretical framework to analytically model the performance of burstable instances, analyze the user selection equilibrium, and maximize the total revenue of a cloud provider. In this section, we first numerically validate this framework and then demonstrate how it can be used to price a public cloud. A Java-based simulator is implemented to simulate the operations of token buckets, regulators, and VMs. The simulations are driven by the Microsoft Azure traces [5]. Released in 2017, these traces are the latest characterization of VM resource utilization in public clouds.

### *A. Validating Our Performance Model*

We validate our performance model (Section II) in this sub-section.

*1) Simulation settings:* The Microsoft Azure traces record CPU utilization of VMs at a time granularity of five minutes. Therefore, the duration of a time slot in our simulations is also set to be five minutes, and a token refers to 1% of the full capacity of a vCPU for five minutes. Five different service class configurations, listed in Table II, are considered in the simulations. We set the token bucket size $b_j$ as the number of tokens earned in 24 hours, as done in Amazon EC2 [7] and Microsoft Azure [8]. At the beginning of the time horizon, every instance is assigned initial tokens for a smooth bootstrap, the amount of which is equivalent to $1/6$ of its token bucket size. Meanwhile, since the average resource volume received per instance is no larger than 20% of a vCPU according to Table II, VMs with an average CPU utilization higher than 20% of a vCPU are excluded from the simulations because they definitely cannot receive their requested resources and are thus not suitable for our burstable instance services. (These VMs may subscribe to traditional static instances due to their high volumes of CPU resources requested.)

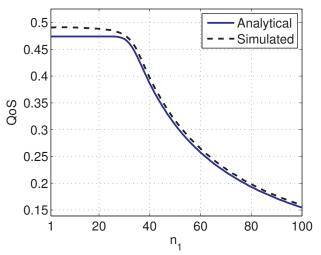
We sort the instance records in chronological order, and randomly select 200 of the first $5,200$ records[7] as samples to estimate the parameter $\delta$ and the distribution of the random variable $\theta \in [1,99] \cap \mathbb{Z}^{+}$. We use these estimates to set our parameters throughout this section. The remaining $5,000$ instance records are used as *testing* data in the simulations in this sub-section. The $\delta$ value and the $\theta$ distribution are respectively obtained by simply counting the number of times that users have resource requests to make and the frequency of appearance of different $\theta$ values in the 200 sample instance records. Our obtained $\delta$ value is 0.9948. Interested readers can refer to Figure 8 in Appendix E-A for the obtained cumulative distribution of $\theta$. We confirm from the traces that $P(\theta = x)$ for random variable $\theta$ is positive at all integral points $x \in [1,99] \cap \mathbb{Z}^{+}$, meaning that our prior assumption in Section II-B holds. We also observe that the distribution of $\theta$ has a long tail, indicating that the users' resource requests are indeed bursty (i.e., varying significantly over time). The $\alpha$ parameter in the QoS metric and the QoS lower bound $\tau$ are set to be 0.9 and 0.1, respectively.

*2) Results:* Our performance model in Section II-B presents the analytical performance of an individual service class given its configuration parameters ($b_j$, $c_j$, and $r_j$) and the number of subscribers ($n_j$). Due to the limited space, we take three of the service classes from Table II, namely, $j = 1$, 3, and 5, as representatives to validate our performance model. Note that all five service class configurations listed in Table II will be considered as we move on to cloud-level simulations with multiple service classes later in this section. We simulate a total period of five days, and play back the workloads in the traces. Our performance models with both the *random selection* and *proportional allocation* schemes will be verified.

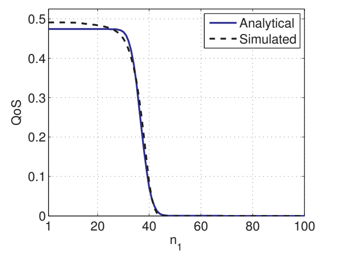
In Figure 2, we show comparisons between our analytical (from Section II) and simulated QoS curves, both obtained by varying the number of users $n_j$ from 1 to 100 for service classes $j = 1, 3$, and 5, with *random selection* and *proportional allocation*, respectively. In the simulated QoS curves, a point corresponding to $n_j$ users shows the average QoS over 25 runs, with $n_j$ instance records randomly drawn from the $5,000$ testing records in each run. Qualitatively, it can be observed from the figure that our analytical curves are close to their simulated counterparts. The average error ratios of our analytical curves to the simulated curves for service class $j = 1$, 3, and 5 is 2.76%, 2.32%, and 0.49% for *random selection*, and 2.96%, 2.92%, and 0.76% for *proportional allocation*, respectively, which are relatively small. Thus, our analytical performance model can both qualitatively and quantitatively well approximate the actual QoS.

*3) Insights:* Next, we elaborate the insights delivered by the QoS curves shown in Figure 2. Under the same service

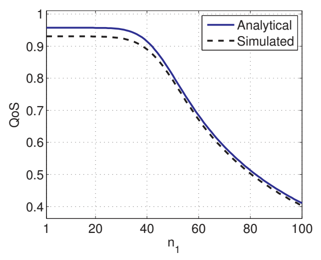
[7]All of these $5,200$ instances start in the first time slot of the time horizon and have durations longer than five days.



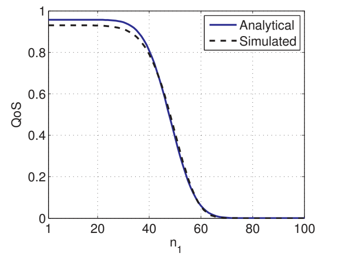
(a) $j = 1$, *random selection*.



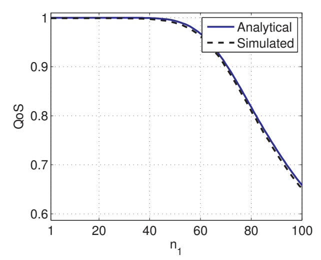
(b) $j = 1$, *proportional allocation*.



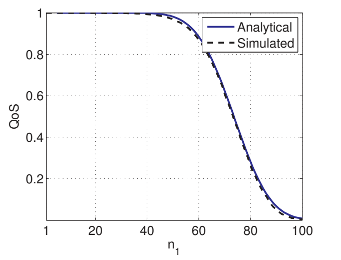
(c) $j = 3$, *random selection*.



(d) $j = 3$, *proportional allocation*.



(e) $j = 5$, *random selection*.



(f) $j = 5$, *proportional allocation*.

Fig. 2: The analytical and simulated QoS curves obtained by varying $n_j$ from 1 to 100 for service classes $j = 1$, 3, and 5, whose parameters are listed in Table II. Both *random selection* and *proportional allocation* are considered. The error ratios of our analytical curves from (a) to (f) are 2.76%, 2.96%, 2.32%, 2.92%, 0.49%, and 0.76% In summary, our analytical performance model can approximate the actual QoS.

class configuration (e.g., $j = 1$), the QoS achieved by *random selection* and *proportional allocation* at $n_j = 1$ is the same. This is because when $n_j = 1$, the examined user has no other peer users to compete with for resources. Meanwhile, the resource capacity at the regulator is always sufficient to accommodate this user's requests (i.e., $c_j \geq \tilde{\theta}_{max}$). Therefore, the QoS for $n_j = 1$ represents the probability that $\alpha$ of a user's requests can traverse the token bucket, and such a QoS is not influenced by the multiplexing scheme at the regulator as long as $c_j \geq \tilde{\theta}_{max}$. On the other hand, with the increase of $n_j$, the QoS achieved by *proportional allocation* deteriorates much faster than that achieved by *random selection*. This is because under the *proportional allocation* scheme, when the total number of requests received by the regulator exceeds the resource capacity, each user gets an equal proportion of his/her requested resources that are received by the regulator. That is to say, if service class $j$'s regulator receives more than $c_j/\alpha$ requests in a particular time slot, none of the users' QoS requirements (i.e., receiving at least $\alpha$ of the user's requested resources) can be fulfilled in this time slot. However, if *random selection* is applied to the same situation, some of the users will be selected to receive their full requested resources, while other users' requests will be rejected. In this case, even if more than $c_j/\alpha$ requests arrive at the regulator, there will still be some users whose QoS requirements can be satisfied.

The observations above imply a *performance-fairness trade-off* behind the multiplexing schemes. When users' QoS requirements cannot be satisfied simultaneously, each user still receives an equal proportion of his/her resource requests that reach the regulator under the *proportional allocation* scheme, although none of the users' received resources can achieve the QoS-required amount (i.e., $\alpha$ of the requested resources). In contrast, users are no longer guaranteed to receive any resources under the *random selection* scheme for this situation. Only some of the users can receive their requested resources with the corresponding QoS requirements satisfied, while the remaining users will be allocated no resources at all.

To further illustrate the fairness of the multiplexing schemes, we calculate the *Gini coefficient*[8] on the ratio of each user's received resources to the requests that the regulator receives for this user in a given time slot. Figure 3 reports the average Gini coefficient with three service class configurations, $j = 1$, 3, and 5, for both *random selection* and *proportional allocation* when we vary $n_j$ from 1 to 100. Since users receive the same proportion of their requests that reach the regulator, the Gini coefficient for *proportional allocation* is always 0. With *random selection*, it can be observed that when the number of users in the service class ($n_j$) is small, the corresponding Gini coefficient is close to 0 because each user receives the full amount of resources that (s)he requests to the regulator most of the time. However, with an increase of $n_j$, the Gini coefficient also increases, as the regulator's resource capacity can no longer satisfy all users. In this case, only a selected group of users are able to receive their requested resources, making the proportions of received resources for different users more diverse. Note that the average resources received by each user over time is the same for the *random selection* and *proportional allocation* schemes, but *random selection* ensures users' requests are occasionally matched.

## *B. Validating Our Equilibrium and Revenue Maximization*

This sub-section validates our equilibrium analysis (Section III) and revenue maximization scheme (Section IV).

*1) Results:* Consider that users' QoS valuations $u_i$ follow a uniform distribution within $(0, 1]$. The CDF at $v_j$ is thus

$$f(v_j) = v_j, \ \ j = 0, 1, \ldots, M. \tag{18}$$

Substituting equation (18) into (10), (11), and (12), we obtain the analytical equilibrium representation.

Two classes of approaches can be applied to solve our revenue maximization problem in Section IV, the general-purpose methods for solving mixed-integer programs [14]

[8]The Gini coefficient is a widely used measure of dispersion on a set of data. A Gini coefficient takes a fractional value within the range [0, 1], where 0 means the values of the elements in the data set are exactly equal to each other, while 1 expresses the maximal inequality among the elements. Details of the Gini coefficient can be found in [26].

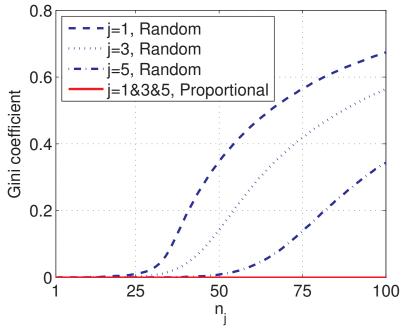
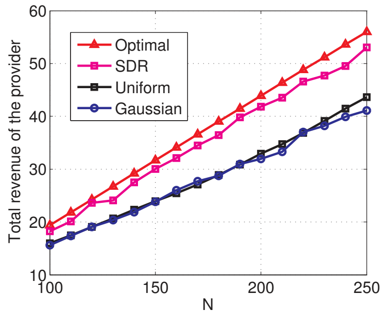
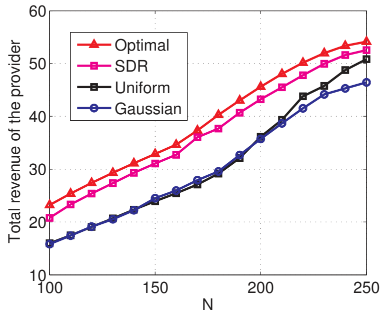


Fig. 3: The Gini coefficients for the six simulated QoS curves in Figure 2.



(a) With *random selection*.



(b) With *proportional allocation*.

Fig. 4: Provider's total revenue under different schemes by varying $N$.

(among which we use the brute-force search algorithm for the simulations in this section), and the SDR-based Algorithm 1 specifically designed based on our problem structure. We will evaluate the performance of both approaches. In the SDR-based approach, a least-squares quadratic approximation is used to approximate $q_j$ in Step 2 of Algorithm 1. Since $q_j$ is non-increasing in $n_j$, we first use a bi-section method to find the maximum $n_j$ that can satisfy the QoS requirement $\tau$, denoted by $n_j^{upper}$, where $q_j\left(n_j\right) < \tau$, $\forall n_j > n_j^{upper}$. Note that $n_j^{upper}$ will also be used later in constructing heuristic benchmarks to be compared with our proposed approaches. Next, we numerically calculate $q_j(n_j)$ for $n_j = 1$, $n_j = n_j^{upper}$, and all the $n_j$ that are integral multiples of 20. For example, if $n_j^{upper} = 90$, we calculate $q_j(n_j)$ for $n_j = 1$, 20, 40, 60, 80, and 90. These numerically calculated $q_j(n_j)$ points are used for the least-squares quadratic approximation. We will demonstrate in the upcoming results that this simple approximation method can provide sufficiently good results. Note that our Algorithm 1 can be combined with any approximation algorithm that returns linear or quadratic approximations of $q_j(n_j)$.

We compare the results derived from the general-purpose method (referred to as the *optimal* approach hereinafter) and SDR-based Algorithm 1 (referred to as the *SDR* approach hereinafter) with two benchmarks: the *uniform* benchmark and the *Gaussian* benchmark, both of which heuristically determine $n_j$, $j \in \mathcal{M}$. Note that $n_0$ users with the lowest QoS valuations $u_i$ do not have incentives to subscribe to service classes; we then set $n_0/N = 0.1$. Among the remaining $N - n_0$ users, the basic idea for the *uniform* benchmark is to admit an equal number of users to each service class. Nevertheless, if $(N - n_0)/M > n_j^{upper}$ for service class $j$, i.e., a strict uniform allocation would lead to infeasible QoS in class $j$, we assign $n_j^{upper}$ users to this service class, and equally assign the remaining users to other service classes with richer resources (i.e., with indices larger than $j$). The *uniform* benchmark is formally presented as Algorithm 3 in Appendix E-B. The *Gaussian* benchmark determines $n_j$, $j \in \mathcal{M} \setminus \{M\}$ sequentially, starting from $j = 1$. To determine an $n_j$, we first draw a random number $v$ from a Gaussian distribution with mean $(N - \sum_{k=0}^{j-1} n_k)/(M - j + 1)$ and standard deviation $(N - \sum_{k=0}^{j-1} n_k)/3(M - j + 1)$. We let $n_j = v$ if $v \le n_j^{upper}$, and $n_j = n_j^{upper}$ otherwise. Finally, $n_M$ is calculated by equation (10). The *Gaussian* benchmark is formally presented as Algorithm 4 in Appendix E-B. By employing the *Gaussian* benchmark, we aim to randomly add some non-linearity to the solution and see if a better performance can be achieved. After vector $\mathbf{n}$ is worked out for the two benchmarks, other decision variables $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{v}$ are then determined by equations (1), (10), and (12) to ensure that they constitute a Nash equilibrium. Other simulation settings stay unchanged from those in Section V-A2. The simulation parameters are properly selected to ensure the feasibility of the revenue maximization problem.

When varying $N$, the total number of users, the corresponding revenues generated by our proposed approaches and the benchmarks for *random selection* and *proportional allocation* are shown in Figure 4a and Figure 4b, respectively. It can be seen from the figures that the *optimal* approach always derives the maximum revenue. Our proposed *SDR* approach is also a good approximation of the *optimal* approach.

*2) Insights:* Table III lists the prices ($p_j$), analytical QoS ($q_j$), numbers of admitted users ($u_j$), and the QoS valuation boundary points ($v_j$) generated by our proposed approaches and the benchmarks for each service class $j$ when $N = 250$. (Due to the limited space, we take the statistics of $N = 250$ as an example.) This table offers us insights into three interesting observations from Figure 4.

First, we elaborate the reasons why our proposed approaches outperform the benchmarks, as shown in Figure 4. It can be observed from Table III that our approaches attach more importance to improving the QoS offered by service classes with richer resources (i.e., with larger indices) by restricting them to fewer users. These users also have higher QoS valuations $u_i$ according to Lemma 2, resulting in higher utilities ($u_i q_j$) being achieved. They are thus willing to pay higher prices, ultimately leading to an increase in the provider's revenue.

The second observation from Figure 4 is that *random selection* derives a slightly higher *optimal* revenue than *proportional allocation* when $N \geq 230$, while the opposite is true when $N < 230$. To understand this better, we additionally list the service-class-wise results obtained by the *optimal* approach for $N = 200$ in Table IV. Note that the price for a service class depends on the differences in the offered QoS between it and its neighboring service classes according to equation (12).

Intuitively, a service class charges more if it "distinguishes" itself more from its lower-level service classes in terms of QoS. Meanwhile, Figure 2 shows that given the same number of users in a service class, *random selection* can offer a better QoS than *proportional allocation*. Therefore, when the total number of users $N$ is not large (e.g., $N = 200$), *proportional allocation* offers worse QoS for low-level service classes (with small indices) than *random selection*. In high-level service classes (with large indices), however, there are still few users and thus little competition among users so that *proportional allocation* can still offer good QoS. In this case, the inter-class QoS differences for *proportional allocation* are higher than those for *random selection*, ultimately leading to a higher total revenue. However, with the increase of $N$, the QoS offered by low-level service classes for *proportional allocation* will reach their lower bounds $\tau$. To accommodate more users (e.g., when $N = 250$), the QoS of high-level service classes must be impaired. On the contrary, *random selection* can still offer high QoS for high-level service classes. In this case, *random selection* produces larger inter-service-class QoS differences and thus a higher revenue.[9]

Our third observation from Figure 4 is that as $N$ increases, the total revenues derived by the benchmarks get closer to the *optimal* revenue for *proportional allocation*, but get farther away for *random selection*. Note that both our proposed approaches and the benchmarks should guarantee $\tau$, the lower bound of the QoS offered by service classes. We define $n_j^{upper}$ as the corresponding maximum number of users that service class $j$ can admit to guarantee $\tau$. Figure 2 shows that $n_j^{upper}$ is lower for *proportional allocation* than it is for *random selection* with the same service class configurations. The maximum number of users that *proportional allocation* can admit under Table II's service class configurations is $\sum_{j=1}^{M} n_j^{upper} = 297$. In contrast, $\sum_{j=1}^{M} n_j^{upper}$ is larger than $1,000$ for *random selection*. Therefore, when $N$ is getting closer to 297, especially within $[220, 250]$, as shown in Figure 4b, the decision space (i.e., the number of feasible combinations) in $\mathbf{n}$ for *proportional allocation* is shrinking, while that for *random selection* is still expanding. Thus, as discussed above, the optimal revenue for proportional allocation increases less than for random selection with the number of users. Expanding the decision space, to the contrary, enlarges the range, and reduces the chances that good performances will be generated by the benchmarks. Take $N = 250$ for *proportional allocation*, as shown in Table III, as an example. The number of users admitted by service class 1 reaches its upper bound $n_1^{upper} = 39$ for both our proposed approaches and the benchmarks, meaning $n_1$ is always at its optimal value.

*3) Impact of $n_0$ on total revenue:* From the provider's perspective, $n_0/N$ can be interpreted as the rejection rate of users. A smaller $n_0/N$ means more users whose QoS valuations satisfy $u_i \in [n_0/N, \gamma]$ will be admitted by service classes at equilibrium. To understand how $n_0$ influences the revenue, we vary $n_0$ with $N$ fixed as 150 and 250, and report the corresponding *optimal* revenues in Figure 5 for *random selection*. Due to the limited space, results for *proportional allocation* are not presented as they are similar to those for *random selection*. In Figure 5, when $n_0/N$ starts to increase from 0, the overall $u_i$ values of the admitted users also increase. As fewer users are admitted, the offered QoS $q_j$ increases for service classes. According to equation (8), higher $u_i q_j$ values leave more room for providers to set higher prices $p_j$, so the corresponding revenue rises. On the other hand, when $n_0/N$ is too high, the number of admitted users becomes

[9]Practically, whether to implement *random selection* or *proportional allocation* depends on the cloud provider's understanding of the market as to what the corresponding parameter $N$ and distribution of $u_i$ will be. For example, *proportional allocation* may attract fewer users than *random selection* due to the lower absolute QoS offered. Interested readers may refer to research on consumer behaviors for this. The aim of this paper is to help providers understand the performance and set prices for burstable instances given the system parameters.

TABLE III: Service-class-wise results for the $N = 250$ case in Figure 4. The result that generates the median revenue over 25 runs for the *Gaussian* scheme is reported.

| $j$ | | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **Random Selection** | | | | | | |
| Optimal | $p_j$ | 0.0142 | 0.3409 | 0.4488 | 0.4768 | 0.4799 |
| | $q_j$ | 0.1419 | 0.7514 | 0.9512 | 0.9935 | 0.9972 |
| | $n_j$ | 109 | 1 | 31 | 38 | 46 |
| | $v_j$ | 0.5360 | 0.5400 | 0.6640 | 0.8160 | 1.0000 |
| SDR | $p_j$ | 0.0184 | 0.2657 | 0.2855 | 0.4136 | 0.4166 |
| | $q_j$ | 0.1841 | 0.7514 | 0.7956 | 0.9919 | 0.9957 |
| | $n_j$ | 84 | 3 | 51 | 39 | 48 |
| | $v_j$ | 0.4360 | 0.4480 | 0.6520 | 0.8080 | 1.0000 |
| Gaussian | $p_j$ | 0.0336 | 0.1071 | 0.1344 | 0.3871 | 0.3917 |
| | $q_j$ | 0.3362 | 0.5950 | 0.6515 | 0.9948 | 1.0000 |
| | $n_j$ | 46 | 50 | 63 | 37 | 29 |
| | $v_j$ | 0.2840 | 0.4840 | 0.7360 | 0.8840 | 1.0000 |
| Uniform | $p_j$ | 0.0344 | 0.1218 | 0.2198 | 0.2878 | 0.3063 |
| | $q_j$ | 0.3437 | 0.6558 | 0.8690 | 0.9752 | 0.9978 |
| | $n_j$ | 45 | 45 | 45 | 45 | 45 |
| | $v_j$ | 0.2800 | 0.4600 | 0.6400 | 0.8200 | 1.0000 |
| **Proportional Allocation** | | | | | | |
| Optimal | $p_j$ | 0.0113 | 0.0187 | 0.2748 | 0.4214 | 0.4483 |
| | $q_j$ | 0.1129 | 0.1418 | 0.7085 | 0.9433 | 0.9771 |
| | $n_j$ | 39 | 49 | 43 | 43 | 51 |
| | $v_j$ | 0.2560 | 0.4520 | 0.6240 | 0.7960 | 1.0000 |
| SDR | $p_j$ | 0.0113 | 0.0293 | 0.2071 | 0.4098 | 0.4847 |
| | $q_j$ | 0.1129 | 0.1835 | 0.5803 | 0.9010 | 0.9928 |
| | $n_j$ | 39 | 48 | 46 | 46 | 46 |
| | $v_j$ | 0.2560 | 0.4480 | 0.6320 | 0.8160 | 1.0000 |
| Gaussian | $p_j$ | 0.0113 | 0.0989 | 0.2710 | 0.2963 | 0.2965 |
| | $q_j$ | 0.1129 | 0.4552 | 0.8574 | 0.9010 | 0.9012 |
| | $n_j$ | 39 | 43 | 38 | 46 | 59 |
| | $v_j$ | 0.2720 | 0.4480 | 0.6320 | 0.8280 | 1.0000 |
| Uniform | $p_j$ | 0.0113 | 0.0551 | 0.1855 | 0.3742 | 0.4619 |
| | $q_j$ | 0.1129 | 0.2840 | 0.5803 | 0.8827 | 0.9908 |
| | $n_j$ | 39 | 46 | 46 | 47 | 47 |
| | $v_j$ | 0.2560 | 0.4400 | 0.6240 | 0.8120 | 1.0000 |

TABLE IV: Service-class-wise results for the $N = 200$ case in Figure 4 by the *optimal* approach.

| $j$ | | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Random Selection | $p_j$ | 0.0176 | 0.3284 | 0.4406 | 0.4656 | 0.4667 |
| | $q_j$ | 0.1758 | 0.7514 | 0.9572 | 0.9975 | 0.9989 |
| | $n_j$ | 88 | 1 | 15 | 34 | 42 |
| | $v_j$ | 0.5400 | 0.5450 | 0.6200 | 0.7900 | 1.0000 |
| Proportional Allocation | $p_j$ | 0.0113 | 0.0198 | 0.4599 | 0.4862 | 0.4871 |
| | $q_j$ | 0.1129 | 0.1418 | 0.9568 | 0.9975 | 0.9987 |
| | $n_j$ | 39 | 49 | 21 | 31 | 40 |
| | $v_j$ | 0.2950 | 0.5400 | 0.6450 | 0.8000 | 1.0000 |

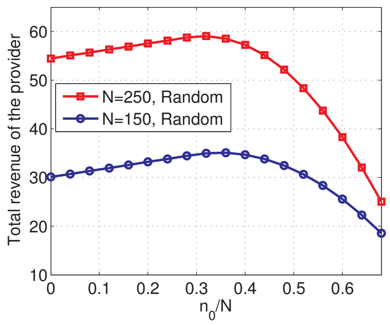
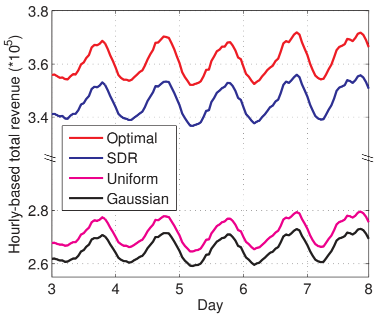
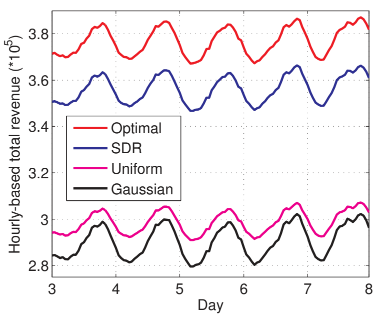


Fig. 5: Optimal total revenues under different $n_0/N$ for *random selection*.



(a) With *random selection*.



(b) With *proportional allocation*.

Fig. 6: Hourly revenue over five days for the case study in Section V-C.

extremely low. Setting higher prices can no longer compensate for the smaller number of users admitted, ultimately leading to a decrease in the total revenue.

## *C. Pricing a Public Cloud: A Use Case Scenario*

In this sub-section, we apply our framework to pricing a public cloud for burstable instance services. The Microsoft Azure traces are used as the VM workloads. Through trace analysis, we find that although VMs are dynamically created and terminated over time, the number of simultaneously running VMs is periodic on a daily basis. Therefore, we regard the workload records from day 1 and day 2 as historical data, which we use to calculate the prices. We then run simulations to evaluate our derived prices using the workload records in a five-day period from day 3 to day 7.

To accommodate the large number of concurrently running VMs in the traces, we duplicate each service class $j$ in Table II 675 times, and refer to such a duplicated service class as a type-$j$ service class. In this case, we have $3,375$ service classes from five types in total. Other parameters stay the same as those in Section V-B. We set $N = 135,000$, corresponding to the peak number of VMs in the system over time. We then partition the service classes into 675 groups, with five different types of service classes and 200 ($= N/675$) users in each group. A group can be implemented using 17 ($= a^{grntd} \cdot N/675 + \sum_{j=1}^{M} c_j$, where $a^{grntd} = 0.01$) vCPUs. Since a group represents a separate set of VMs, a VM's received QoS depends only on the behaviors of other VMs within the same group. We can thus calculate the prices for our proposed approaches and the benchmarks within a group at equilibrium, the same as in Section V-B. Since groups are homogeneous, all type-$j$ service classes essentially have the same $p_j$, $q_j$, $n_j$, and $v_j$. The simulation parameters are selected to ensure that we eventually get non-trivial results, which are not extreme cases and can gain us insights.

The cloud assigns VMs to service classes upon their creations and removes them upon their terminations. When a new VM $i$ needs to be created, we first check its $u_i$ to decide which service class type it should go to. The VM is then assigned to the service class that has the minimum number of active VMs within this type, i.e., VMs that are currently running. As the QoS and prices are designed with regard to the peak demand, the actual number of VMs in a service class is smaller than the designed number most of the time. In this case, VMs can receive higher actual QoS than that guaranteed by our pricing approach during off-peak periods. When an existing VM terminates, we simply remove it from its service class. We regard our derived prices as the payment of an active VM for a time slot's (i.e., five minutes') duration. For example, a VM subscribing to service class $j$ for an hour should pay $20p_j$ in total. We plot the hourly-based revenues in the time horizon under both our proposed approaches and the benchmarks in Figure 6. Our *optimal* approach is shown to yield the best revenues for both *random selection* and *proportional allocation*. Our *SDR* approach also generates the second-best revenues as a good approximation.

## VI. Related Work

Existing works on burstable instances fall into two classes. On the infrastructure level, the first class of works studies how the CPU credit mechanism works. Through extensive measurements, Leitner *et al.* [27] verify that the CPU credit mechanism works as advertised by cloud providers (e.g., Table I). Wang *et al.* [13] further point out that this mechanism essentially follows a token bucket model [15], [16], [17]. This finding has motivated us to model burstable instances and analytically study the performance. To bridge the gap between the performance of burstable instances and their commercial operation, we continue to study how a cloud provider should price burstable instances for the maximum revenue. To the best of our knowledge, we are the first to study optimal pricing for burstable instances.

The second class of existing works focuses on use cases of burstable instances. Wang *et al.* [10] present the deployment of backup services on burstable instances, while Baarzi *et al.* [28] present another deployment of web servers and in-memory cache. Also, both Yan *et al.* [29] and Ali *et al.* [30], [31] discuss how to shape the CPU resource utilization of applications to make full use of the initial CPU credits assigned to burstable instances. This class of works on the application level is different from ours. From a cloud provider's perspective on the infrastructure level, we have no control over the behaviors of applications, but just take and process their resource requests.

Burstable instances and the correspondingly introduced resource provisioning mechanism have been attracting more and more attention from the research community. While burstable instances were initially designed for computation resources, Park *et al.* [32] extend them to storage services. In their proposed system, I/O credits, which follow the same philosophy as CPU credits of burstable instances, take the role to regulate users' received storage resources.

Similar techniques to those employed in our work, such as token bucket models and optimal pricing, have been used to address different problems in the literature. For example, token bucket models have been extensively adopted to regulate data traffic [15], [16], [17]. Other works have priced service classes with differentiated QoS levels in data networks [23], [33]. However, due to different system dynamics and characteristics, these results cannot be directly applied to our burstable instance scenario. Similarly, the distinct features of burstable instances compared to traditional static cloud instances prevent existing models on cloud pricing (e.g., [34] and [35]) from being applied to our scenario.

Some early works have proposed alternative resource provisioning ideas to tackle bursty workloads in clouds. Wang *et al.* [36] propose to aggregate the bursty workloads in a cloud broker for cost savings to users. The broker reserves cheap long-term resources from the cloud provider and profits from the aggregation. A similar notion has been studied in [37], but the brokerage strategy follows a different business model and system characteristics to ours. The model we study stems from current practices in the industry. Another stream of works has investigated, from the applications' perspective, how resource requests should be made via proactive prediction [38], [39] or online algorithmic decision processes [40], [41], while our work focuses on how resource requests already made by users can be accommodated by a cloud provider.

## VII. Conclusion

This paper presents a framework to analytically model the performance of burstable instances given service class configurations (Section II), characterize users' selections of service classes at the Nash equilibrium (Section III), and maximize the provider's total revenue by finding the optimal prices at equilibrium (Section IV). We validate our framework via trace-driven simulations. The results show that our performance model can estimate the QoS received by burstable instances with an average error ratio lower than 3%, and our revenue maximization scheme can increase the provider's revenue compared to heuristic methods (Section V).

As the first to study burstable instances from a theoretical perspective, we regard this work as an initial framework that captures the fundamental features of burstable instances. To extend the work, more diverse settings can be integrated into our framework. For example, we can consider a hybrid cloud that offers both static and burstable instances. By allocating different proportions of the resources to the two types of instances, users' selection behaviors and the cloud provider's optimal revenue could be further studied. Another direction is to continue to study the theoretical bound of our proposed SDR-based algorithm for revenue maximization.

## References

[1] Y. Jiang, M. Shahrad, D. Wentzlaff, D. H. K. Tsang, and C. Joe-Wong, "Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2019.

[2] "Amazon EC2 pricing." [Online]. Available: https://aws.amazon.com/ec2/pricing/

[3] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM Symp. Cloud Comput.*, 2012, pp. 7:1–7:13.

[4] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Netw.*, vol. 29, no. 2, pp. 56–61, 2015.

[5] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. ACM Symp. Operating Syst. Principles*, 2017, pp. 153–167.

[6] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via VM multiplexing," in *Proc. Int. Conf. Autonomic Comput.*, 2010, pp. 11–20.

[7] "CPU credits and baseline performance for burstable performance instances." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html

[8] "Microsoft Azure: B-series burstable virtual machine sizes." [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/b-series-burstable

[9] "Google Cloud Platform: Shared-core machine types." [Online]. Available: https://cloud.google.com/compute/docs/machine-types#sharedcore

[10] C. Wang, B. Urgaonkar, A. Gupta, G. Kesidis, and Q. Liang, "Exploiting spot and burstable instances for improving the cost-efficacy of in-memory caches on the public cloud," in *Proc. European Conf. Comput. Syst.*, 2017, pp. 620–634.

[11] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 793–804.

[12] M. Shahrad, C. Klein, L. Zheng, M. Chiang, E. Elmroth, and D. Wentzlaff, "Incentivizing self-capping to increase cloud utilization," in *Proc. ACM Symp. Cloud Comput.*, 2017, pp. 52–65.

[13] C. Wang, B. Urgaonkar, N. Nasiriani, and G. Kesidis, "Using burstable instances in the public cloud: Why, when and how?" *Proc. ACM on Measurement and Anal. Comput. Syst.*, vol. 1, no. 1, p. 11, 2017.

[14] D. Li and X. Sun, *Nonlinear Integer Programming*. Springer Science & Business Media, 2006, vol. 84.

[15] C. Courcoubetis and V. A. Siris, "Managing and pricing service level agreements for differentiated services," in *Proc. IEEE/ACM Int. Symp. Quality Service*, 1999, pp. 165–173.

[16] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 337–348, 2007.

[17] F. M. F. Wong, C. Joe-Wong, S. Ha, Z. Liu, and M. Chiang, "Improving user QoE for residential broadband: Adaptive traffic management at the network edge," in *Proc. IEEE/ACM Int. Symp. Quality Service*, 2015, pp. 105–114.

[18] M. Björkqvist, N. Gautam, R. Birke, L. Y. Chen, and W. Binder, "Optimizing for tail sojourn times of cloud clusters," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 156–167, 2018.

[19] W. Zheng, M. Zhou, L. Wu, Y. Xia, X. Luo, S. Pang, Q. Zhu, and Y. Wu, "Percentile performance estimation of unreliable IaaS clouds and their cost-optimal capacity decision," *IEEE Access*, vol. 5, pp. 2808–2818, 2017.

[20] Z. Huang and D. H. Tsang, "M-convex VM consolidation: Towards a better VM workload consolidation," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 415–428, 2016.

[21] R. G. Gallager, "Finite state Markov chains," in *Discrete Stochastic Processes*. Springer, 1996, pp. 103–147.

[22] C. Joe-Wong, S. Sen, and S. Ha, "Offering supplementary network technologies: Adoption behavior and offloading benefits," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 355–368, 2014.

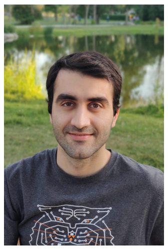
[23] S. Shakkottai, R. Srikant, A. Ozdaglar, and D. Acemoglu, "The price of simplicity," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 7, 2008.

[24] Z.-Q. Luo, W.-K. Ma, A. M.-C. So, Y. Ye, and S. Zhang, "Semidefinite relaxation of quadratic optimization problems," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 20–34, 2010.

[25] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. 16th Annu. ACM Symp. Theory Comput.*, 1984, pp. 302–311.

[26] C. Gini, "Concentration and dependency ratios," *Rivista di Politica Economica*, vol. 87, pp. 769–792, 1997.
[27] P. Leitner and J. Scheuner, "Bursting with possibilities–an empirical study of credit-based bursting cloud instance types," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput.*, 2015, pp. 227–236.
[28] A. F. Baarzi, T. Zhu, and B. Urgaonkar, "Burscale: Using burstable instances for cost-effective autoscaling in the public cloud," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 126–138.
[29] F. Yan, L. Ren, D. J. Dubois, G. Casale, J. Wen, and E. Smirni, "How to supercharge the Amazon t2: Observations and suggestions," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2017, pp. 278–285.
[30] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Cedule: A scheduling framework for burstable performance in cloud computing," in *Proc. IEEE Int. Conf. Auton. Comput.*, 2018, pp. 141–150.
[31] ——, "It's not a sprint, it's a marathon: Stretching multi-resource burstable performance in public clouds," in *Proc. Int. Middleware Conf. Ind. Track*, 2019, pp. 36–42.
[32] H. Park, G. R. Ganger, and G. Amvrosiadis, "More IOPS for less: Exploiting burstable storage in public clouds," in *Proc. 12th USENIX Workshop Hot Topics Cloud Comput.*, 2020.
[33] A. Odlyzko, "Paris metro pricing for the internet," in *Proc. ACM Conf. Electronic Commerce*, 1999, pp. 140–147.
[34] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 2015, pp. 71–84.
[35] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 158–171, 2013.
[36] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. IEEE Int. Conf. Distributed Computing Systems*, 2013, pp. 400–409.
[37] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang, "On the viability of a cloud virtual service provider," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 44, no. 1, pp. 235–248, 2016.
[38] J. Tai, J. Zhang, J. Li, W. Meleis, and N. Mi, "ARA: Adaptive resource allocation for cloud computing environments under bursty workloads," in *Proc. IEEE Int. Performance Comput. and Commun. Conf.*, 2011, pp. 1–8.
[39] D. J. Dubois and G. Casale, "Performance prediction for burstable cloud resources," in *Proc. EAI Int. Conf. Performance Evaluation Methodologies and Tools*, 2017, pp. 217–218.
[40] H. Zhang, G. Jiang, K. Yoshihira, and H. Chen, "Proactive workload management in hybrid cloud computing," *IEEE Trans. Netw. Serv. Manag.*, vol. 11, no. 1, pp. 90–100, 2014.
[41] N. Morris, C. Stewart, L. Chen, R. Birke, and J. Kelley, "Model-driven computational sprinting," in *Proc. EuroSys Conf.*, 2018, pp. 38:1–38:13.
[42] Y. Jiang, Z. Huang, and D. H. K. Tsang, "On power-peak-aware scheduling for large-scale shared clusters," *IEEE Trans. Big Data*, vol. 6, no. 2, pp. 412–426, 2020.
[43] H. Xu and B. Li, "Reducing electricity demand charge for data centers with partial execution," in *Proc. ACM Int. Conf. Future Energy Syst.*, 2014, pp. 51–61.
[44] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI Conf. Artificial Intell.*, 2017.
[45] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L.-C. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Tech. Mag.*, vol. 14, no. 2, pp. 44–52, 2019.

**Yuxuan Jiang** (Member, IEEE) received his Ph.D. degree from The Hong Kong University of Science and Technology in 2019 and his B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China in 2013. He was a visiting scholar at Carnegie Mellon University, working with Prof. Carlee Joe-Wong. His research interests lie in the broad area of networking and distributed systems, with a focus on resource management in cloud/edge computing and big data systems.



**Mohammad Shahrad** (Member, IEEE) is an incoming Assistant Professor of Electrical and Computer Engineering at The University of British Columbia. He received his M.A. and Ph.D. degrees from Princeton University in 2016 and 2020, respectively. Prior to that, he received his B.Sc. in Electrical Engineering from Sharif University of Technology in 2014. Dr. Shahrad's research aims to improve the efficiency of public cloud systems through better resource management and enhanced vertical integration. He has experience studying and building real cloud systems, for which he won the USENIX Community Award in 2020.

**David Wentzlaff** (Member, IEEE) is currently an Associate Professor with the Electrical Engineering Department, Princeton University. His research interests include parallel computer architecture, architectures for cloud computing, and biodegradable computing systems. He has received the NSF CAREER award, the DARPA Young Faculty Award, the AFOSR Young Investigator Prize, and the Princeton E. Lawrence Keyes Faculty Advancement Award. Wentzlaff received the master's and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology.

**Danny H.K. Tsang** (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the Moore School of Electrical Engineering at the University of Pennsylvania, U.S.A., in 1989. Upon graduation, he joined the Department of Computer Science at Dalhousie University in Canada. He later joined the Department of Electronic and Computer Engineering at The Hong Kong University of Science and Technology (HKUST) in 1992 and is now a Professor in the department. He has also served as Leader of the Internet of Things Thrust Area at HKUST (Guangzhou Campus) since 2020. He was a Guest Editor for IEEE Journal of Selected Areas in Communications' special issue on Advances in P2P Streaming Systems, an Associate Editor for Journal of Optical Networking published by the Optical Society of America, and a Guest Editor for IEEE Systems Journal. He currently serves as Technical Editor for IEEE Communications Magazine. He was nominated to become an IEEE Fellow in 2012 and an HKIE Fellow in 2013. During his leave from HKUST in 2000-2001, Dr. Tsang assumed the role of Principal Architect at Sycamore Networks in the United States. He was responsible for the network architecture design of Ethernet MAN/WAN over SONET/DWDM networks. He invented the 64B/65B encoding (US Patent No.: US 6,952,405 B2) and contributed it to the proposal for Transparent GFP in the T1X1.5 standard that was advanced to become the ITU G.GFP standard. The coding scheme has now been adopted by International Telecommunication Union (ITU)'s Generic Framing Procedure recommendation GFP-T (ITU-T G.7041/Y.1303)) and Interfaces for the Optical Transport Network (ITU-T G.709). His current research interests include cloud computing, edge computing, NOMA networks and smart grids.

**Carlee Joe-Wong** (Member, IEEE) is an Assistant Professor of Electrical and Computer Engineering at Carnegie Mellon University. She received her A.B., M.A., and Ph.D. degrees from Princeton University in 2011, 2013, and 2016, respectively. Dr. Joe-Wong's research is in optimizing networked systems, particularly on applying machine learning and pricing to data and computing networks. From 2013 to 2014, she was the Director of Advanced Research at DataMi, a startup she co-founded from her Ph.D. research on mobile data pricing. She has received a few awards for her work, including the ARO Young Investigator Award in 2019, the NSF CAREER Award in 2018, and the INFORMS ISS Design Science Award in 2014.

## Appendix A
## Examples on Translating Our Infrastructure-Level QoS to the Performance of User-Deployed Applications

Similar to other studies on IaaS clouds, our QoS metric quantifies the infrastructure-level resource availability. With the control of low-level resources, such as the CPU, IaaS cloud users deploy their applications onto the infrastructure (i.e., the VMs) provided by the IaaS clouds. Since an IaaS cloud provider has neither the knowledge of what applications the users are running on their VMs, nor the control of these applications, the QoS metric for an IaaS cloud does not directly capture the performance of user-deployed applications (e.g., the completion time of a Hadoop job [42]). Having the full knowledge of his/her deployed applications, the user may translate the infrastructure-level QoS to the application-level performance to get a better understanding of the quality of an IaaS cloud service. Although such a translation is out of the scope of our work, which focuses on the resource provisioning of an IaaS cloud, we still shed some light on how this translation may be done for the two sample applications mentioned in Section I, namely, hot standbys and periodically updating machine learning models.

Hot standbys and updating machine learning models belong to a class of applications that tolerate partial execution [43]. In other words, although these applications may receive only a part of their requested resources from the underlying burstable instances, they can run with these partial resources and achieve a certain performance.

The function of a hot standby is to execute as many workloads as possible for the main service when the main service is down, in order to reduce the amount of pending workloads when the main service recovers. From our infrastructure-level QoS, the user, by making resource requests, can have an idea of how many of them will be accepted. Although the user may only receive a part of his/her requested resources, his/her hot standby can still use these resources to process some of the workloads submitted to the main service, and the application-level performance improves when more workloads can be processed by the hot standby to mitigate the workload of the main service when it is recovered.

In the example of training a machine learning model, its performance can be quantified by the accuracy of the trained model. In many cases, the training process is iterative and the more iterations that the training process undergoes, the higher the accuracy of the trained model will be. Therefore, the actual amount of resources that the user receives in the infrastructure level can translate to the number of iterations that the machine learning model will be trained with, and finally be translated to the accuracy of the trained model. For example, if a user receives 90% of his/her requested resources, (s)he can complete 90% of the desired iterations and get within less than 10% of the targeted training error.[10]

We should acknowledge that in real-world systems, users' applications may not tolerate partial execution (e.g., with the job completion time as the performance indicator). Therefore, the QoS translation from the infrastructure level to the application level may be more complicated than what we have stated above.

[10]The marginal improvement in the model accuracy usually decreases with the number of iterations increases [44], [45].

## Appendix B
## Proof of Proposition 1 in Section II

*Proof.* The proposition defines the transition probability from state $d$ in the current time slot to state $h$ in the next time slot under the following three different scenarios, with the visualizations of their state transition probabilities $P_{d \to h}$ shown in Figure 7.

(i) When $r_j \leq d \leq \theta_{max} - 1$, we first examine the case where the user has resource requests in the current time slot. Depending on $\theta$, the number of requests, there are two possible sub-scenarios for the state transition:

- When $\theta \leq d$, the number of currently available tokens, $d$, is sufficient to satisfy all the requests. Therefore, $\theta$ tokens will be consumed in the current time slot, and $r_j$ more tokens will be accumulated in the beginning of the next time slot, so the Markov chain will transit to state $h = d - \theta + r_j$. We re-write the above equation with regard to $\theta$ as $\theta = d + r_j - h$. Therefore, with probability $\delta \cdot P(\theta = d + r_j - h)$, the Markov chain transits from state $d$ to state $h$, with the range of $h$ as $r_j \leq h \leq d + r_j - 1$.
- When $\theta > d$, $d$ out of $\theta$ requests will be fulfilled by the currently available tokens, with the rest of the requests being discarded. After accumulating $r_j$ new tokens in the beginning of the next time slot, the Markov chain will transit to state $h = r_j$. The probability for such a transition is $\delta \cdot \sum_{k=d+1}^{\theta_{max}} P(\theta = k)$.

If the user does not have resource requests in the current time slot with probability $1 - \delta$, the Markov chain will transit from state $d$ to state $h = d + r_j$ in the next time slot, because no tokens are consumed in the current time slot, and $r_j$ tokens are accumulated in the beginning of the next time slot. Meanwhile, the Markov chain cannot transit to states $h > d + r_j + 1$. To sum up, the state transition probabilities when $r_j \leq d \leq \theta_{max} - 1$ are written as

$$P_{d \to h} = \begin{cases} \delta \cdot \sum_{k=d}^{\theta_{max}} P(\theta = k) & h = r_j, \\ \delta \cdot P\left(\theta = d + r_j - h\right) & r_j + 1 \leq h \\ & \leq d + r_j - 1, \\ 1 - \delta & h = d + r_j, \\ 0 & \text{otherwise.} \end{cases}$$

(ii) When $\theta_{max} \leq d \leq b_j - r_j$, we always have sufficient tokens to satisfy requests if there are such requests sent by the user with probability $\delta$. In this case, The Markov chain will transit from state $d$ to state $h = d - \theta + r_j$ with probability $\delta \cdot P(\theta = d + r_j - h)$. If there are no requests sent by the user in the current time slot with probability $1 - \delta$, the Markov chain will transit from state $d$ to state $h = d + r_j$. The Markov chain cannot transit to states $h < d + r_j - \theta_{max}$ or $h > d + r_j - 1$. In summary, the state transition probabilities when $\theta_{max} \leq d \leq b_j - r_j$ are written as

$$P_{d\to h} = \begin{cases} \delta \cdot P\left(\theta = d + r_j - h\right) & d + r_j - \theta_{max} \le h \\ & \le d + r_j - 1, \\ 1 - \delta & h = d + r_j, \\ 0 & \text{otherwise.} \end{cases}$$

(iii) When $b_j - r_j + 1 \le d \le b_j$, we examine the case where the user has resource requests. Two sub-scenarios are considered:

- When $\theta \ge d+r_j-b_j$, $\theta$ tokens are consumed in the current time slot, after which, $r_j$ tokens are newly accumulated in the next time slot. Therefore, the Markov chain will transit from state $d$ to state $h = d-\theta+r_j$ with probability $\delta \cdot P(\theta = d + r_j - h)$.
- When $\theta < d + r_j - b_j$, after consuming $\theta$ tokens, the number of remaining tokens in the token bucket is larger than $b_j - r_j$. Note that we can only accumulate up to $b_j$ tokens in the token bucket. Therefore, the corresponding transition probability from state $d$ to state $b_j$ is $\delta \cdot \sum_{k=1}^{d+r_j-b_j} P(\theta = k)$.

If the user does not have resource requests in the current time slot with probability $1 - \delta$, the Markov chain will also transit to state $b_j$, observing the token bucket size. Meanwhile, the Markov chain cannot transit to states $h < d+r_j-\theta_{max}$. In summary, the state transition probabilities when $b_j - r_j + 1 \le d \le b_j$ are written as

$$P_{d\to h} = \begin{cases} \delta \cdot P\left(\theta = d + r_j - h\right) & d + r_j - \theta_{max} \\ & \le h \le b_j - 1, \\ \delta \cdot \displaystyle\sum_{k=1}^{d+r_j-b_j} P\left(\theta = k\right) + (1-\delta) & h = b_j, \\ 0 & \text{otherwise.} \end{cases}$$

□

## Appendix C
## Proofs in Section III

### A. Proof of Lemma 1

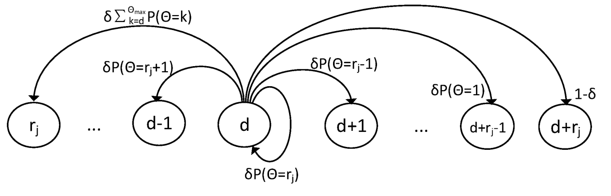
*Proof.* For any user $i$, we have $w_{i,j} - w_{i,k} = u_i\left(q_j - q_k\right) - \left(p_j - p_k\right) > 0$, so all users will prefer service class $j$ to service class $k$ at equilibrium, with $n_k = 0$. □

### B. Proof of Corollary 1

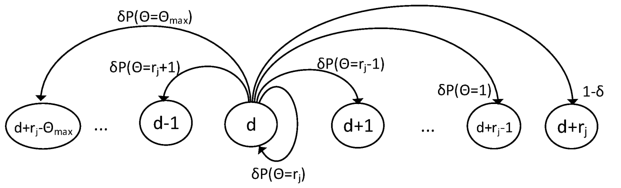
*Proof.* The proof is similar to that of Lemma 1. For any user $i$, we have $w_{i,j} - w_{i,k} = u_i\left(q_j - q_k\right) - \left(p_j - p_k\right) > 0$, so all users will prefer service class $j$ to service class $k$ at equilibrium, with $n_k = 0$. □

### C. Proof of Lemma 2

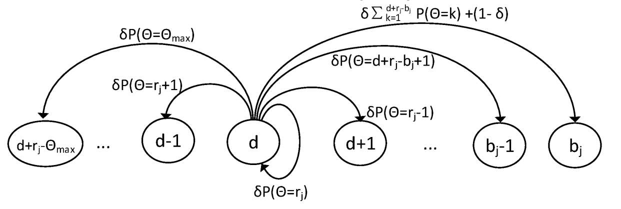
*Proof.* We prove the lemma by contradiction. Suppose $\eta(k) < \eta(i)$, which means $w_{k,\eta(k)} \ge w_{k,\eta(i)}$ according to the equilibrium definition in equation (9). Note that $\eta(k) < \eta(i)$ leads to $q_{\eta(k)} \le q_{\eta(i)}$ according to our non-decreasing sorting in the indices of the service classes. However, if $q_{\eta(k)} = q_{\eta(i)}$, since



(a) $r_j \le d \le \theta_{max} - 1$



(b) $\theta_{max} \le d \le b_j - r_j$



(c) $b_j - r_j + 1 \le d \le b_j$

Fig. 7: Visualization of state transition probability $P_{d\to h}$ from a certain state $d$. (a), (b), and (c) correspond to the three scenarios in Proposition 1.

$u_k > u_i$, $\eta(k) \ge \eta(i)$ must hold. As a result, when $\eta(k) < \eta(i)$, we should have strictly $q_{\eta(k)} < q_{\eta(i)}$. Therefore,

$$\begin{aligned} w_{k,\eta(i)} - w_{k,\eta(k)} &= (u_k q_{\eta(i)} - p_{\eta(i)}) - (u_k q_{\eta(k)} - p_{\eta(k)}) \\ &> (u_i q_{\eta(i)} - p_{\eta(i)}) - (u_i q_{\eta(k)} - p_{\eta(k)}) \\ &= w_{i,\eta(i)} - w_{i,\eta(k)} \\ &\ge 0. \end{aligned}$$

The above inference shows that $w_{k,\eta(i)} > w_{k,\eta(k)}$, which contradicts our initial assumption that $w_{k,\eta(k)} \ge w_{k,\eta(i)}$. □

### D. Proof of Corollary 2

*Proof.* We prove the corollary by contradiction. Suppose $p_1 = 0$, and meanwhile, user $i$ does not have an incentive to subscribe to any of the service classes. In this case, $w_{i,j} < 0$, $\forall j \in \mathcal{M}$ according to equation (9). However, $w_{i,1} = u_i q_1 - p_1 = u_i q_1 \ge 0$, which contradicts our assumption that user $i$ does not have an incentive to subscribe to any service class. □

### E. Proof of Proposition 2

*Proof.* Necessity: From Lemma 2, a necessary condition for a Nash equilibrium, we learn the monotonicity in users' service class selections. Users with higher QoS valuations $u_i$ will subscribe to service classes with higher QoS levels (i.e., higher indices) at equilibrium. Specifically, for each service class $j \in \mathcal{M} \setminus \{M\}$, $v_j$ is a turning point for user $i$'s service class selection. If $u_i \ge v_j$, the user will subscribe to service class $j+1$, which means $v_j q_{j+1} - p_{j+1} \geq v_j q_j - p_j$. If $u_i < v_j$, the user will subscribe to service class $j$, which means $\lim_{u_i \to v_j^-} \left(u_i q_j - p_j\right) \geq \lim_{u_i \to v_j^-} \left(u_i q_{j+1} - p_{j+1}\right)$. Therefore, at the turning point $v_j$, we should have

$$v_j q_j - p_j = v_j q_{j+1} - p_{j+1},\ \forall j \in \mathcal{M} \setminus \{M\}. \tag{19}$$

The same notion holds for boundary point $v_0$, so that

$$v_0 q_1 - p_1 = 0. \tag{20}$$

We can thus recursively write equations (19) and (20) with regard to $p_j$ from $j = 1$ to $j = M-1$ as equation (12). As a result, equation (12) holds at the Nash equilibrium.

Sufficiency: Consider a user $i$ with his/her QoS valuation as $u_i$. If $u_i < v_0$, according to the definitions of $n_j$ and $v_j$ in equation (12), this user will have no incentive to subscribe to any service class because all the service classes will deliver negative rewards to this user. As shown in equation (9), this is by definition the Nash equilibrium and the sufficiency is thus proved for this particular case.

We continue to focus on cases where $u_i \geq v_0$. According to the definitions of $n_j$ and $v_j$ in equation (12), suppose this user should subscribe to a particular service class $j$, where $u_i \in [v_{j-1}, v_j]$ if $j = M$ and $u_i \in [v_{j-1}, v_j)$ otherwise. We prove the sufficiency by contradiction. Suppose $p_j$, $q_j$, $v_j$, and $n_j$, $j \in \mathcal{M}$, satisfy equation (12) but do not constitute a Nash equilibrium. In other words, user $i$ will subscribe to service class $j'$, where $j' \neq j$, at the Nash equilibrium. According to the definition of a Nash equilibrium, as shown in equation (9), we should have

$$w_{i,j} - w_{i,j'} = \left(u_i q_j - p_j\right) - \left(u_i q_{j'} - p_{j'}\right) < 0. \tag{21}$$

In what follows, we respectively delve into the two cases where $j < j'$ and $j > j'$.

*(i)* When $j < j' \leq M$, we have $u_i < v_j$. We can then get equations (22a) to (22k) shown on the next page, where equation (22c) is obtained by substituting equation (12) into equation (22b) and equations (22d), (22f), (22h), and (22k) are obtained by $u_i < v_j < v_k$, $\forall k > j$ and $q_k \leq q_{j'}$, $\forall k < j'$. From equations (22a) to (22k), we know that $w_{i,j} - w_{i,j'} \geq 0$, which contradicts our prior assumption in equation (21) that $w_{i,j} - w_{i,j'} < 0$.

*(ii)* When $1 \leq j' < j$, we have $v_{j-1} \leq u_i$. We can then get equations (23a) to (23k) shown on the next page, where equation (23c) is obtained by substituting equation (12) into equation (23b) and equations (23d), (23f), (23h), and (23k) are obtained by $u_i \geq v_j > v_k$, $\forall k < j$ and $q_k \geq q_{j'}$, $\forall k > j'$. From equations (23a) to (23k), we know that $w_{i,j} - w_{i,j'} \geq 0$, which contradicts our prior assumption in equation (21) that $w_{i,j} - w_{i,j'} < 0$.

□

## Appendix D
## Details of Steps 3 and 4 in Algorithm 1

### *A. Constructing a Semidefinite Relaxed Problem in Step 3*

Let vector $\mathbf{s} = [\mathbf{p}^T\ \mathbf{q}^T\ \mathbf{n}^T\ \mathbf{v}^T]^T$ be the concatenated vector of the decision variables in the optimization problem. We can write the optimization problem after the completion of Step 2 in Algorithm 1 in the following form:

(inhomogeneous QCQP)

$$\begin{aligned}
\underset{\mathbf{s}}{\text{maximize}} \quad & \mathbf{s}^T \mathbf{G}^{(13)} \mathbf{s} \\
\text{subject to} \quad & \mathbf{s}^T \mathbf{G}_j^{(1)} \mathbf{s} + 2\mathbf{s}^T \mathbf{g}_j^{(1)} = \beta_j^{(1)},\ \forall j \in \mathcal{M}, \\
& \mathbf{s}^T \mathbf{G}_j^{(10)} \mathbf{s} + 2\mathbf{s}^T \mathbf{g}_j^{(10)} = \beta_j^{(10)},\ \forall j \in \mathcal{M}, \\
& \mathbf{s}^T \mathbf{G}_j^{(12)} \mathbf{s} + 2\mathbf{s}^T \mathbf{g}_j^{(12)} = 0,\ \forall j \in \mathcal{M}, \\
& \mathbf{s}^T \mathbf{G}_j^{(14)} \mathbf{s} + 2\mathbf{s}^T \mathbf{g}_j^{(14)} \leq 0,\ \forall j \in \mathcal{M} \setminus \{M\}, \\
& \mathbf{s}^T \mathbf{G}^{(15)} \mathbf{s} + 2\mathbf{s}^T \mathbf{g}^{(15)} \geq \tau, \\
& \mathbf{s}^T \mathbf{G}_j^{(17)} \mathbf{s} + 2\mathbf{s}^T \mathbf{g}_j^{(17)} \geq 0,\ \forall j \in \mathcal{M},
\end{aligned}$$

where $\mathbf{G}_j^{(k)}$ is the coefficient matrix of the quadratic term in constraint $(k)$ with index $j$, while $\mathbf{g}_j^{(k)}$ is the coefficient vector of the linear term and $\beta_j^{(k)}$ is the constant term. Particularly, we have no $j$ index in the objective function and constraint (15), so $\mathbf{G}^{(13)}$ and $\mathbf{G}^{(15)}$ directly represent the coefficient matrices of the quadratic terms in the objective function (13) and constraint (15), respectively. We refer to constraints (1) and (10) in the above inhomogeneous QCQP formulation as their convex forms after the approximations in Step 2 of Algorithm 1. To apply SDR to the inhomogeneous QCQP formulation, we first need to transform it to a homogeneous QCQP.[11] To this end, we first introduce an additional decision variable $t$ that satisfies $t^2 = 1$. Let vector $\mathbf{z} = [\mathbf{s}^T\ t]^T$ be the new decision variable vector. To merge the linear term $2\mathbf{s}^T\mathbf{g}_j^k$ into the quadratic term $\mathbf{s}^T\mathbf{G}_j^{(k)}\mathbf{s}$, we introduce a new coefficient matrix $\mathbf{H}_j^{(k)}$ for each constraint in the inhomogeneous QCQP formulation so that

$$\mathbf{H}_j^{(k)} = \begin{bmatrix} \mathbf{G}_j^{(k)} & \mathbf{g}_j^{(k)} \\ \mathbf{g}_j^{(k)T} & 0 \end{bmatrix}.$$

We can therefore re-write the optimization problem with $\mathbf{z}$ as the decision variable vector:

(homogeneous QCQP)

$$\begin{aligned}
\underset{\mathbf{z}}{\text{maximize}} \quad & \mathbf{z}^T \mathbf{H}^{(13)} \mathbf{z} \\
\text{subject to} \quad & \mathbf{z}^T \mathbf{H}_j^{(1)} \mathbf{s} = \beta_j^{(1)},\ \forall j \in \mathcal{M}, \\
& \mathbf{z}^T \mathbf{H}_j^{(10)} \mathbf{z} = \beta_j^{(10)},\ \forall j \in \mathcal{M}, \\
& \mathbf{z}^T \mathbf{H}_j^{(12)} \mathbf{z} = 0,\ \forall j \in \mathcal{M}, \\
& \mathbf{z}^T \mathbf{H}_j^{(14)} \mathbf{z} \leq 0,\ \forall j \in \mathcal{M} \setminus \{M\}, \\
& \mathbf{z}^T \mathbf{H}^{(15)} \mathbf{z} \geq \tau, \\
& \mathbf{z}^T \mathbf{H}_j^{(17)} \mathbf{z} \geq 0,\ \forall j \in \mathcal{M} \\
& \mathbf{z}^T \mathbf{H}^{(t)} \mathbf{z} = 1,
\end{aligned}$$

where $\mathbf{H}^{(t)}$ is a matrix with only one non-zero entry valued 1 at the bottom right. The constraint associated with $\mathbf{H}^{(t)}$ guarantees that $t^2 = 1$. Denote $\tilde{\mathbf{z}} = [\tilde{\mathbf{s}}^T\ \tilde{t}]^T$ as the optimal solution to the homogeneous QCQP formulation. If $\tilde{t} = 1$, $\tilde{\mathbf{s}}$ is also an optimal solution to the inhomogeneous QCQP

[11] According to the definition in [24], a homogeneous QCQP is a QCQP with solely quadratic terms in both its objective function and constraints.

$$\begin{aligned}
w_{i,j} - w_{i,j'} &= \left(u_i q_j - p_j\right) - \left(u_i q_{j'} - p_{j'}\right) && \text{(22a)}\\
&= \left(u_i q_j - u_i q_{j'}\right) + \left(p_{j'} - p_j\right) && \text{(22b)}\\
&= u_i \left(q_j - q_{j'}\right) + \left(v_j \left(q_{j+1} - q_j\right) + \sum_{k=j+2}^{j'} v_{k-1} \left(q_k - q_{k-1}\right)\right) && \text{(22c)}\\
&\geq v_j \left(q_j - q_{j'}\right) + \left(v_j \left(q_{j+1} - q_j\right) + \sum_{k=j+2}^{j'} v_{k-1} \left(q_k - q_{k-1}\right)\right) && \text{(22d)}\\
&= v_j \left(q_{j+1} - q_{j'}\right) + \left(v_{j+1} \left(q_{j+2} - q_{j+1}\right) + \sum_{k=j+3}^{j'} v_{k-1} \left(q_k - q_{k-1}\right)\right) && \text{(22e)}\\
&\geq v_{j+1} \left(q_{j+1} - q_{j'}\right) + \left(v_{j+1} \left(q_{j+2} - q_{j+1}\right) + \sum_{k=j+3}^{j'} v_{k-1} \left(q_k - q_{k-1}\right)\right) && \text{(22f)}\\
&= v_{j+1} \left(q_{j+2} - q_{j'}\right) + \left(v_{j+2} \left(q_{j+3} - q_{j+2}\right) + \sum_{k=j+4}^{j'} v_{k-1} \left(q_k - q_{k-1}\right)\right) && \text{(22g)}\\
&\geq \ldots\ldots\\
&\geq v_{j'-2} \left(q_{j'-2} - q_{j'}\right) + \left(v_{j'-2} \left(q_{j'-1} - q_{j'-2}\right) + v_{j'-1} \left(q_{j'} - q_{j'-1}\right)\right) && \text{(22h)}\\
&= v_{j'-2} \left(q_{j'-1} - q_{j'}\right) + v_{j'-1} \left(q_{j'} - q_{j'-1}\right) && \text{(22i)}\\
&\geq v_{j'-1} \left(q_{j'-1} - q_{j'}\right) + v_{j'-1} \left(q_{j'} - q_{j'-1}\right) && \text{(22j)}\\
&= 0. && \text{(22k)}
\end{aligned}$$

---

$$\begin{aligned}
w_{i,j} - w_{i,j'} &= \left(u_i q_j - p_j\right) - \left(u_i q_{j'} - p_{j'}\right) && \text{(23a)}\\
&= \left(u_i q_j - u_i q_{j'}\right) + \left(p_{j'} - p_j\right) && \text{(23b)}\\
&= u_i \left(q_j - q_{j'}\right) + \left(v_{j-1} \left(q_{j-1} - q_j\right) + \sum_{k=j'+1}^{j-1} v_{k-1} \left(q_{k-1} - q_k\right)\right) && \text{(23c)}\\
&\geq v_{j-1} \left(q_j - q_{j'}\right) + \left(v_{j-1} \left(q_{j-1} - q_j\right) + \sum_{k=j'+1}^{j-1} v_{k-1} \left(q_{k-1} - q_k\right)\right) && \text{(23d)}\\
&= v_{j-1} \left(q_{j-1} - q_{j'}\right) + \left(v_{j-2} \left(q_{j-2} - q_{j-1}\right) + \sum_{k=j'+1}^{j-2} v_{k-1} \left(q_{k-1} - q_k\right)\right) && \text{(23e)}\\
&\geq v_{j-2} \left(q_{j-1} - q_{j'}\right) + \left(v_{j-2} \left(q_{j-2} - q_{j-1}\right) + \sum_{k=j'+1}^{j-2} v_{k-1} \left(q_{k-1} - q_k\right)\right) && \text{(23f)}\\
&= v_{j-2} \left(q_{j-2} - q_{j'}\right) + \left(v_{j-3} \left(q_{j-3} - q_{j-2}\right) + \sum_{k=j'+1}^{j-3} v_{k-1} \left(q_{k-1} - q_k\right)\right) && \text{(23g)}\\
&\geq \ldots\ldots\\
&\geq v_{j'+1} \left(q_{j'+2} - q_{j'}\right) + \left(v_{j'+1} \left(q_{j'+1} - q_{j'+2}\right) + v_{j'} \left(q_{j'} - q_{j'+1}\right)\right) && \text{(23h)}\\
&= v_{j'+1} \left(q_{j'+1} - q_{j'}\right) + v_{j'} \left(q_{j'} - q_{j'+1}\right) && \text{(23i)}\\
&\geq v_{j'} \left(q_{j'+1} - q_{j'}\right) + v_{j'} \left(q_{j'} - q_{j'+1}\right) && \text{(23j)}\\
&= 0. && \text{(23k)}
\end{aligned}$$

formulation. Otherwise, if $\tilde{t} = -1$, $-\tilde{\mathbf{s}}$ is an optimal solution to the inhomogeneous QCQP formulation.

Let $\mathbf{Z} = \mathbf{z}\mathbf{z}^T$. Observe that

$$\begin{aligned}\mathbf{z}^T\mathbf{H}_j^{(k)}\mathbf{z} &= \mathrm{Tr}\left(\mathbf{z}^T\mathbf{H}_j^{(k)}\mathbf{z}\right)\\ &= \mathrm{Tr}\left(\mathbf{H}_j^{(k)}\mathbf{z}\mathbf{z}^T\right)\\ &= \mathrm{Tr}\left(\mathbf{H}_j^{(k)}\mathbf{Z}\right),\end{aligned}$$

where $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix. The above relationship allows us to equivalently transform the homogeneous QCQP formulation to a rank-constrained semidefinite program (SDP) as follows:

(rank-constrained SDP)

$$\begin{aligned}\underset{\mathbf{Z}}{\text{maximize}}\quad & \mathrm{Tr}\left(\mathbf{H}^{(13)}\mathbf{Z}\right)\\ \text{subject to}\quad & \mathrm{Tr}\left(\mathbf{H}_j^{(1)}\mathbf{Z}\right) = \beta_j^{(1)},\ \forall j \in \mathcal{M},\\ & \mathrm{Tr}\left(\mathbf{H}_j^{(10)}\mathbf{Z}\right) = \beta_j^{(10)},\ \forall j \in \mathcal{M},\\ & \mathrm{Tr}\left(\mathbf{H}_j^{(12)}\mathbf{Z}\right) = \beta_j^{(12)},\ \forall j \in \mathcal{M},\\ & \mathrm{Tr}\left(\mathbf{H}_j^{(14)}\mathbf{Z}\right) \le 0,\ \forall j \in \mathcal{M} \setminus \{M\},\\ & \mathrm{Tr}\left(\mathbf{H}^{(15)}\mathbf{Z}\right) \ge \tau,\\ & \mathrm{Tr}\left(\mathbf{H}_j^{(17)}\mathbf{Z}\right) \ge 0,\ \forall j \in \mathcal{M},\\ & \mathrm{Tr}\left(\mathbf{H}^{(t)}\mathbf{Z}\right) = 1,\\ & \mathrm{Rank}(\mathbf{Z}) = 1.\end{aligned}$$

Note that the last constraint regulates the rank of the $\mathbf{Z}$ matrix to be 1, so that we are able to recover the original optimal $\mathbf{z}^*$ vector from the optimal $\mathbf{Z}^*$ matrix. Observe that the only non-convex constraint in the rank-constrained SDP formulation is the rank constraint. Therefore, the core idea of SDR is to relax the rank constraint so that the remaining convex optimization problem can be solved efficiently. In other words, after removing the rank constraint, we have successfully constructed the semidefinite relaxed formulation to be solved in Step 3 of Algorithm 1.

The major drawback of removing the rank constraint is that the rank of the obtained optimal solution $\mathbf{Z}^*$ is no longer guaranteed to be 1. Therefore, we may not be able to directly recover a feasible $\mathbf{z}$ vector using the equation $\mathbf{Z} = \mathbf{z}\mathbf{z}^T$. We will demonstrate in Appendix D-B the detailed recovery approach for a feasible solution to the original revenue maximization problem in Section V.

## *B. Recovering Feasible Solutions to the Original Problem in Step 4*

In this sub-section, we elaborate step 4 in Algorithm 1 on how to recover a feasible solution to the original revenue maximization problem from the optimal solution to the semidefinite relaxed problem. Our proposed recovery approach is summarized as Algorithm 2.

The core idea of the recovery algorithm is the eigenvalue approximation approach [24]. In the beginning of the algorithm, we construct an initial solution from the largest eigenvalue and its corresponding eigenvector of matrix $\tilde{\mathbf{Z}}$ in step 1. Afterwards, we construct feasible $\mathbf{n}$, $\mathbf{q}$, $\mathbf{v}$, and $\mathbf{p}$ sequentially. Specifically, in steps 2 to 8, we first construct a feasible $\mathbf{n}$ vector by proportionally rounding the $\hat{\mathbf{n}}$ vector with regard to $N$ so that constraint (10) can be satisfied. Since we use ceiling functions during the proportional rounding process in step 4, if vector $\hat{\mathbf{n}}$ after proportional rounding is still not feasible, the only possibility is that the summation of all its entries exceeds $N - n_0$. (Note that the decision variable vector $\mathbf{n}$ does not include $n_0$, which is a provider-specified input argument to the optimization algorithm. Thus, if we recursively write constraint (10), the relationship that $\sum_{j=1}^{M} n_j = N - n_0$ can be derived.) We thus deduct the values of some entries in $\mathbf{n}$ to construct a feasible $\hat{\mathbf{n}}$ vector in step 7. In steps 9 to 13, we check whether the derived $n_j$, $j \in \mathcal{M} \setminus \{M\}$ can satisfy the minimum QoS requirement $\tau$ by comparing $n_j$ with $n_j^{upper}$. If $n_j > n_j^{upper}$, we need to equally assign the difference $n_j^{upper} - n_j$ to the service classes $k : j < k \le M$ that have richer resources, and then truncate $n_j$ down to $n_j^{upper}$. With such operations, we can guarantee that $n_j$, $j \in \mathcal{M} \setminus \{M\}$ are feasible. Note that $n_M \le n_M^{upper}$ is also guaranteed as the parameters that we select for simulations ensures that feasible solutions exist for the revenue maximization problem. Now we have obtained a feasible $\mathbf{n}$ vector. We further construct $\mathbf{q}$, $\mathbf{v}$, and $\mathbf{p}$ sequentially according to their dependency constraints in step 14.

**Algorithm 2** Feasible Solution Recovery in Step 4 of Algorithm 1.

```
Input: Optimal solution $\tilde{\mathbf{Z}}$ to the semidefinite relaxed formulation in Step 3 of Algorithm 1 and all the inputs to Algorithm 1.
Output: Feasible solution $\mathbf{p}$, $\mathbf{q}$, $\mathbf{n}$, and $\mathbf{v}$ to the original revenue maximization problem in Section IV.
 1: Construct vector $\hat{\mathbf{z}} = \sqrt{\lambda}\tilde{\rho}$, where $\lambda$ is the largest eigenvalue of $\tilde{\mathbf{Z}}$, while $\tilde{\rho}$ is its corresponding eigenvector.
 2: Extract the $\hat{\mathbf{n}}$ vector from the $\hat{\mathbf{z}}$ vector, and ceil the $\hat{\mathbf{n}}$ vector: $\hat{\mathbf{n}} = \lceil\hat{\mathbf{n}}\rceil$.
 3: if the summation of all the entries in vector $\hat{\mathbf{n}}$ does not equal $N - n_0$, i.e., sum($\hat{\mathbf{n}}$) $\neq N - n_0$, then
 4:     Proportionally scale vector $\hat{\mathbf{n}}$ as $\hat{n}_i = \lceil(N - n_0) \cdot \hat{n}_i/\mathrm{sum}(\hat{\mathbf{n}})\rceil$ for all the entries in vector $\hat{\mathbf{n}}$, where $\hat{n}_i$ denotes the $i$th entry in vector $\hat{\mathbf{n}}$.
 5: end if
 6: if sum($\hat{\mathbf{n}}$) $\neq N - n_0$ then
 7:     Deduct the last sum$(\hat{\mathbf{n}}) - (N - n_0)$ entries in $\hat{\mathbf{n}}$ whose values are larger than 1 by 1.
 8: end if
 9: for $j = 1, 2, ..., M - 1$ do
10:     if $n_j > n_j^{upper}$ then
11:         Amortize the difference $\left(n_j^{upper} - n_j\right)$ equally to $n_k$, $j < k \le M$. Then let $n_j = n_j^{upper}$.
12:     end if
13: end for
14: Let $\mathbf{n} = \hat{\mathbf{n}}$. Construct $\mathbf{q}$ from $\mathbf{n}$ according to equation (1). Construct $\mathbf{v}$ from $\mathbf{n}$ according to equation (10). Construct $\mathbf{p}$ from $\mathbf{q}$ and $\mathbf{v}$ according to equation (12).
15: return vectors $\mathbf{p}$, $\mathbf{q}$, $\mathbf{n}$, and $\mathbf{v}$.
```

## Appendix E
## Supplemental content to Section V

### A. Supplemental Figure to Section V

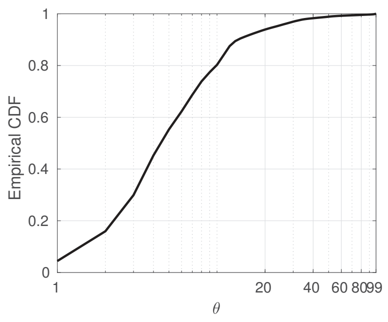


Fig. 8: The empirical cumulative distribution of $\theta$ derived from the traces.

The cumulative distribution of $\theta$ that we derive from the Microsoft Azure traces [5] is shown in Figure 8. It can be observed that $\theta$ has a long-tail distribution. Most of the time, the amount of resources that a user requests is small, but occasionally, (s)he may request a large amount of resources (i.e., having a bursty workload). Meanwhile, we confirm from the traces that $P(\theta = x)$ for random variable $\theta$ is positive at all integral points $x \in [1, 99] \cap \mathbb{Z}^+$, meaning our prior assumption in Section II-B that $P(\theta = x) > 0$, $x \in [1, \theta_{max}] \cap \mathbb{Z}^+$ holds.

### B. *Formal Presentation of the* Uniform *and* Gaussian *Benchmarks in Section V*

In this sub-section, we formally present the *uniform* benchmark in Section V as Algorithm 3, and the *Gaussian* benchmark as Algorithm 4.

**Algorithm 3** The *Uniform* Benchmark in Section V.

```
Input: The same as the input to Algorithm 1.
Output: Feasible solution n.
1: for $j = 1, 2, ..., M-1$ do
2:     if $\lfloor \frac{N-\sum_{k=0}^{j-1} n_k}{M-j+1} \rfloor \le n_j^{upper}$ then
3:         $n_j = \lfloor \frac{N-\sum_{k=0}^{j-1} n_k}{M-j+1} \rfloor$.
4:     else
5:         $n_j = n_j^{upper}$.
6:     end if
7: end for
8: $n_M = N - \sum_{k=0}^{M-1} n_k$.
9: return vector $\mathbf{n} = [n_1, n_2, ..., n_M]^T$.
```

**Algorithm 4** The *Gaussian* Benchmark in Section V.

```
Input: The same as the input to Algorithm 1.
Output: Feasible solution n.
1: for $j = 1, 2, ..., M-1$ do
2:     Draw a random number $\nu$ from a Gaussian distribution with mean $\lfloor \frac{N-\sum_{k=0}^{j-1} n_k}{M-j+1} \rfloor$ and standard deviation $\lfloor \frac{N-\sum_{k=0}^{j-1} n_k}{3(M-j+1)} \rfloor$.
3:     if $\nu \le n_j^{upper}$ then
4:         $n_j = \nu$.
5:     else
6:         $n_j = n_j^{upper}$.
7:     end if
8: end for
9: $n_M = N - \sum_{k=0}^{M-1} n_k$.
10: return vector $\mathbf{n} = [n_1, n_2, ..., n_M]^T$.
```