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

Yuxuan Jiang*, Mohammad Shahrad†, David Wentzlaff†, Danny H.K. Tsang*, and Carlee Joe-Wong‡
*The Hong Kong University of Science and Technology, †Princeton University, ‡Carnegie Mellon University
Emails: *{yjiangad, eetsang}@ust.hk, †{mshahrad, wentzlaf}@princeton.edu, ‡cjoewong@andrew.cmu.edu

***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 in different service classes, identify key performance metrics, and derive the 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 the 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 a public cloud.**

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

I. Introduction

To reduce costs for cloud tenants, today's cloud providers offer various pricing schemes, such as on-demand pricing, spot pricing, and reserved pricing [1]. Under these pricing schemes, however, tenants always obtain virtual machines (VMs) provisioned with static amounts of resources; for example, 1 virtual CPU (vCPU) and 2 GB memory. On the other hand, empirical studies [2]–[5] 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: it is lower than 35% on average according to a Google cluster trace study [3], and lower than 20% for 60% of the VMs according to a Microsoft Azure trace study [4]. These observations imply that tenants' costs can be further reduced by time-varying resource provisioning. In particular, VMs can benefit from *bursts*, which mean receiving a high volume of resources for a short period of time, in exchange for fewer resources most of the time. To this end, a new class of VMs, named *burstable* instances, has recently been introduced by a number of cloud providers, including Amazon EC2 [6], Google Cloud Engine [7], and Microsoft Azure [8]. In this paper, we approach burstable instances *from a theoretical perspective*, and present the first unified framework to model, analyze, and optimize the operation of burstable instances. We show that cloud providers can use this framework to understand the performance of burstable instances, and dramatically increase their total revenue compared to other heuristic pricing methods.

TABLE I: Samples of Amazon EC2 burstable instances [6].

Instance type	CPU credits earned per hour	Maximum CPU credits buffered	Resource volume (vCPUs)	
			Maximum	Mean
t2.nano	3	72	1	0.05
t2.micro	6	144		0.1
t2.small	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. A CPU credit provides 100% of the full capacity of a vCPU for a time slot's duration (e.g., 1 minute in Amazon EC2 [9]). 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 1 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 *t2.nano* 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 requiring large amounts of resources occasionally. 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. Periodic workloads, such as periodic machine learning in online social networks [11], transaction updates [12], and statistical calculations [13], are also suitable use cases 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. 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 [3]. 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 [14]. 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 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 refer 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 probability that a VM is successfully allocated the resources that it requests, i.e., how well the 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 then model the dynamics of CPU credits as a token bucket regulation mechanism [15]. Meanwhile, we model the resource multiplexing scheme that burstable instance services adopt, and finally derive an analytical QoS representation.

From an individual user's perspective, which service class should (s)he select to selfishly maximize his/her reward? We proceed to study a cloud that offers burstable instances with multiple service classes configured by different CPU credit parameters and prices. A rational user favors a service class that offers higher QoS with less 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. The service class selection decision is complicated, however, by the following dynamics: When a user switches between two service classes, the total numbers of subscribers to the two service classes change. Since the resources within a service class are shared by all the subscribers to this class, the change in the number of subscribers leads to a change of the QoS offered by the service class, possibly prompting other users to adjust their service class selections as well. In Section III, we analytically derive users' service class selections at the equilibrium of these user decision dynamics.

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 the 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. Therefore, given the service class configurations, a cloud provider can set the prices leveraging prior knowledge of the equilibrium on users' corresponding subscription decisions, so as to maximize its total revenue at equilibrium. In Section IV, we study the problem of obtaining such optimal prices for the cloud provider.

Our answers to these three questions constitute a framework to model, analyze, and optimize burstable instance services. In Section V, we numerically validate our framework using real-world traces, and show that it drastically improves 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, thus developing an analytical framework to analyze burstable instances. 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, detailed proofs of lemmas, corollaries and propositions are presented in our online technical report [16].

II. Performance Modeling

In this section, we first introduce the system model of a 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. 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

[1]Over-commitment in clouds means the resources allocated to the VMs on a server can exceed the server's actual capacity, and the VMs are expected not to fully utilize their reserved resources simultaneously [3]. Therefore, VMs may not always receive the full resources that they demand.

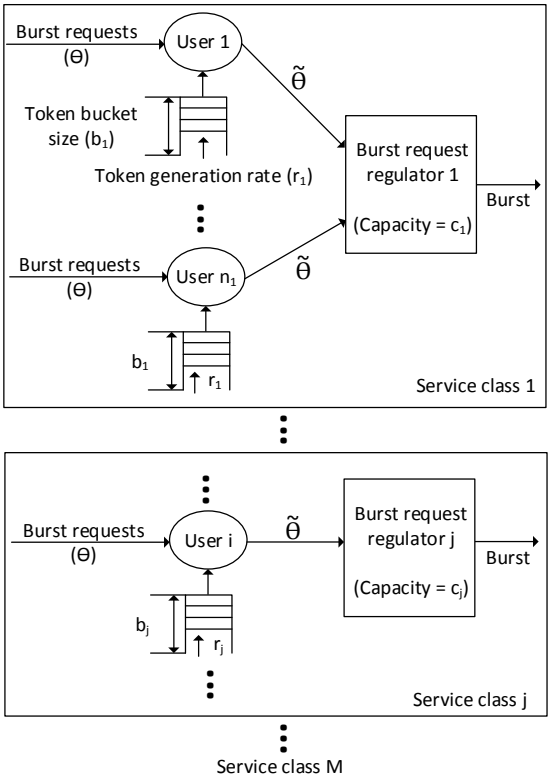


Fig. 1: The system model of M service classes in a cloud. A user corresponds to an instance (equivalently, a VM) in the cloud, and maintains a token bucket. Users subscribing to the same service class share a regulator that multiplexes the burstable resources. After burst requests are made, these requests get the corresponding tokens and proceed to the regulator if the number of tokens in the token bucket is no smaller than the number of burst requests; otherwise, the requests are discarded. The regulator guarantees that the burstable resources allocated to instances do not exceed the capacity of each service class.

a service class. Suppose service class $j \in \mathcal{M} = \{1, 2, ..., M\}$ has n_j subscribed users. A dedicated token bucket is employed to model the CPU credit mechanism for each individual user [15]. In our model, we convert CPU credits to a more fine-grained unit named tokens. A token stands for the smallest resource unit that can be scheduled in the system with one time slot's duration. In the beginning of a time slot, r_j tokens are generated to the token bucket of each user subscribing to service class j. As an example to demonstrate 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 1 minute. A *t2.nano* instance shown in Table I receives 0.05 CPU credits per time slot, equivalent to 5 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[2] of resources to maintain its underlying essentials, such as the operating system. Therefore, even if an instance's resource requests are rejected in a time slot (for example, due to too many simultaneous resource requests from peer instances), the instance will not halt due to insufficient resources received. Our model falls back to the basic burstable instance design presented in Table I when $a^{grntd} = 0$. Since guaranteed resources are always available and 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, an instance may make burst requests for additional resources, which are referred to as burstable resources hereinafter. The resource multiplexing strategy considered in this paper is to reserve c_j token units of burstable resources for the requests from all subscribers of service class j. To sum up, a service class j can be characterized by three parameters, b_j, c_j, and r_j, which are set by the cloud provider.

In a time slot after tokens are accumulated, a user can send requests for burstable resources. Each burst request is for one token unit. For example, if the user wants 5% of a vCPU, with one token standing for 1% of a vCPU, a batch of five burst requests will be sent. Upon arrival, each burst request checks if there is an available token in the token bucket; if all requests can be matched with available tokens, the requests proceed to the burst request regulator and the corresponding tokens are deducted from the user's token bucket. Otherwise, the requests will be discarded. The user may send new requests in the coming time slots if (s)he still demands burstable resources.

The regulator of service class j receives burst requests from users, and admit users' requests according to the following policy: the regulator uniformly at random selects a user and admits his/her requests, until the resource capacity is used up or all users' requests are admitted.

B. Quantifying Users' QoS

In this paper, we are interested in the analytical form of a user's received QoS when subscribing to a service class. For simplicity, we assume that users are homogeneous. (We will demonstrate in our trace-driven simulations in Section V that our model can still handle realistic scenarios with heterogeneous users.) The probability that a user has at least one burst request in a time slot is denoted by δ. The number of burst requests that a user sends in a time slot, given that (s)he has burst requests, is a random variable $\theta \in [1, \theta_{max}] \cap \mathbb{Z}^+$. For example, if a token stands for 1% of a vCPU, with the maximum resource volume as 1 vCPU and guaranteed resources as 2% of a vCPU ($a^{grntd} = 2$) for each instance, an instance can request up to 98% of a vCPU as its burstable resources ($\theta_{max} = 98$). Denote the probability that θ takes the value x by $P(\theta = x)$. The distribution of θ can be estimated from historical data of CPU utilization.

The QoS that user $i \in \mathcal{N} = \{1, 2, ..., N\}$ subscribing to service class j receives is defined as the probability that the user can successfully receive all the burstable resources that (s)he requests, denoted by q_j. (We assume users are not strategic in the timing of their burst requests.) For simplicity, we do not consider partial fulfillment of burst requests. In other words, a user's burst requests in a time slot either all receive tokens and proceed to the regulator, or are all rejected.

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

In order for a user's burst requests to be satisfied in a given time slot, two conditions have to be met: (i) sufficient tokens are available in the user's token bucket, and (ii) the regulator admits the user's burst request. The QoS equation is thus

$$q_j = s_j^{tkn} \cdot s_j^{rgltr}, \tag{1}$$

where s_j^{tkn} is the probability that the first condition is satisfied, and s_j^{rgltr} is the probability that the second condition is satisfied given that the first condition has already been satisfied. In what follows, we respectively derive s_j^{tkn} and s_j^{rgltr}.

1) Derivation of s_j^{tkn}: 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 burst requests are sent and processed. In this case, the token bucket has at least r_j tokens and thus we have $b_j - r_j + 1$ states. Denote state $d = r_j, r_j + 1, ..., b_j$ as the $(d - r_j + 1)$th state of the Markov chain with d tokens in the bucket. We observe that normally $b_j > 2\theta_{max}$ holds.[3] Therefore, we suppose this relationship also holds for this paper. Note that even if $b_j > 2\theta_{max}$ does not hold, our derived results can be easily generalized by using the same methodology. The state transition probabilities of the Markov chain are given by Proposition 1.

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

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

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

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

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

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

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

The Markov chain is recurrent and aperiodic, so it is ergodic [17]. Denote the steady state probability of state d by π_j^d. We can obtain π_j^d by solving the balance equation. Consequently, s_j^{tkn} is given by

$$s_j^{tkn} = \sum_{d=r_j}^{\theta_{max}} P\left(\theta \le d\right) \pi_j^d + \sum_{d=\theta_{max}+1}^{b_j} \pi_j^d, \tag{2}$$

where each term of the summations on the right-hand side represents that there are enough tokens available in the token bucket to accommodate d arriving burst requests.

2) Derivation of s_j^{rgltr}: Since we assume users are homogeneous, it suffices to derive s_j^{rgltr} from the perspective of a single user. Let $\tilde{\theta} \in [1, \tilde{\theta}_{max}] \cap \mathbb{Z}^+$ denote the number of burst requests that the regulator receives from a user, given that the user has enough tokens for its burst requests to reach the regulator. The range of $\tilde{\theta}$ is the same as that of θ (i.e., $\tilde{\theta}_{max} = \theta_{max}$). The probability that $\tilde{\theta}$ takes the value x is

$$P(\tilde{\theta} = x) = \begin{cases} \frac{P(\theta = x)}{s_j^{tkn}} & 1 \le x \le r_j, \\ \frac{P(\theta = x) \cdot \sum_{d=x}^{b_j} \pi_j^d}{s_j^{tkn}} & r_j + 1 \le x \le \theta_{max}. \end{cases} \tag{3}$$

Equation (4) on top of the next page gives the analytical form of s_j^{rgltr}, i.e., the probability that the examined user's θ burst requests, which have already traversed the token bucket, are admitted by the regulator. To derive s_j^{rgltr}, we recall the regulator's selection policy: The regulator uniformly at random selects a user to admit. If there is not enough residual capacity to admit the user currently under consideration, the regulator does not admit any more users, either. Therefore, we first need to know the number of other users who also have burst requests reaching the regulator, denoted by k in Equation (4), with $\binom{n_j-1}{k} \left(\delta s_j^{tkn}\right)^k \left(1 - \delta s_j^{tkn}\right)^{n_j-1-k}$ as its probability of occurrence. Let $\tilde{\theta_h}$ be the number of burst requests received by the regulator from peer user h. The probability that the examined user is the hth user ($1 \le h \le k+1$) to be selected by the regulator is $1/(k+1)$, and the probability that this user is admitted as the hth-picked user is $P(\theta + \sum_{l=1}^{h-1} \tilde{\theta}_l \le c_j)$.

Substituting Equations (2), (3), and (4) into (1), we obtain the analytical form of q_j. In what follows, when referring to Equation (1), we mean its complete representation after all the above substitutions. Since b_j, c_j, and r_j are pre-configured parameters and θ follows a known probability distribution, the offered QoS q_j of service class j depends only on n_j, the number of users subscribing to it.

III. Equilibrium Analysis

Given a performance model for users' received QoS from each service class, in this section, we further study users' service class selections. Suppose service class $j \in \mathcal{M}$ charges each of its subscribers a price p_j. Meanwhile, we let each user $i \in \mathcal{N}$ specify a coefficient u_i that represents his/her valuation of the received QoS; the user will therefore harvest a utility of $u_i q_j$ by subscribing to service class j. We also assume that each user's u_i value lies on a continuum within the range $(0, \gamma]$, where γ is a parameter. Let $f(x)$ be the cumulative distribution

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

$$s_j^{rgltr} = \sum_{k=0}^{n_j-1} \binom{n_j-1}{k} \left(\delta s_j^{tkn}\right)^k \left(1-\delta s_j^{tkn}\right)^{n_j-1-k} \left(\frac{1}{k+1}\sum_{h=1}^{k+1} P\left(\theta + \sum_{l=1}^{h-1}\tilde{\theta}_l \le c_j\right)\right) \quad (4)$$

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

$$w_{i,j} = u_i q_j - p_j. \quad (5)$$

We are interested in how user i will select the service class, denoted by $\eta(i)$, which yields the maximum reward $w_{i,j}$, $j \in \mathcal{M}$. Without loss of generality, we suppose the indices of service classes are sorted in a non-decreasing order according to the QoS that they offer (i.e., $q_j \le q_k$, $\forall j < k$). Note that the QoS q_j that service class j offers depends on the actual number of its subscribers n_j. Therefore, this service class index ordering may change as users switch among service classes.

In this section, we focus on deriving the **Nash equilibrium** of users' service class selections. An advantage of our following equilibrium analysis is that it makes no assumption on the exact form of users' QoS representation; the QoS may be as specified in Equation (1), or take another form. Thus, our equilibrium results can be generalized to other QoS models. When a Nash equilibrium is reached, each user should receive more reward from his/her selected service class than from any other service classes when all other users' selections remain unchanged, which can be represented as

$$w_{i,\eta(i)} \ge \max\left\{w_{i,j}, 0\right\}, \forall j \in \mathcal{M}\backslash\{\eta(i)\}. \quad (6)$$

Note that a rational user always selects a service class that delivers a non-negative reward to him/her. Therefore, the reward $w_{i,\eta(i)}$ offered by service class $\eta(i)$ should be non-negative. Practically, users can report their QoS valuations u_i to the cloud provider. The provider then makes service class subscription recommendations to its users by computing the user selection equilibrium. Then users can subscribe to their best service classes, so that their rewards are maximized.

In the rest of this section, we characterize the Nash equilibrium properties. Our first result finds necessary conditions on the prices charged by service classes:

Lemma 1. *Consider two service classes, j and k, where $q_j \ge q_k$. If $p_j < p_k$, no user has an incentive to subscribe to service class k at equilibrium.*

Lemma 1 indicates that a service class that offers a relatively lower QoS to its users should also charge a lower price. Otherwise, the service class will have no subscribers at equilibrium. Additionally, we derive the conditions for service classes that offer equal QoS in the following corollary.

Corollary 1. *For two service classes, j and k, where $q_j = q_k$, $p_j = p_k$ should hold. Otherwise, one of the service classes will have no users at equilibrium.*

Corollary 1 suggests that if multiple service classes offer the same QoS to users, they should charge the same price. In this paper, we assume that the prices are set according to Lemma 1 and Corollary 1. Otherwise, some service classes will become redundant and can be eliminated because rational users have no incentive to subscribe to them, and the cloud provider will derive no profit from them. Without loss of generality, 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 $\eta(i) < \eta(k)$. In the next lemma, we qualitatively illustrate the relationship between users' QoS valuations and their service class selections.

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 $\eta(k) \ge \eta(i)$.*

Next we derive a necessary 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 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 $u_i \in [v_{M-1}, v_M]$ will subscribe to service class M. Users with $u_i \in (0, v_0)$ do not have incentives to subscribe to any service class. (If $p_1 = 0$, then $v_0 = 0$ according to Corollary 2.) 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. Therefore, the relationship between the number of subscribers n_j of service class j and the corresponding boundary points v_{j-1} and v_j is

$$n_j = N\left(f\left(v_j\right) - f\left(v_{j-1}\right)\right),\ j \in \mathcal{M}. \quad (7)$$

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

$$n_0 = N f(v_0). \quad (8)$$

Now we are ready to derive users' selections of service classes at equilibrium as shown in Proposition 2.

Proposition 2. *At equilibrium, p_j can be written by*

$$p_j = v_0 q_1 + \sum_{k=2}^{j} v_{k-1}\left(q_k - q_{k-1}\right),\ \forall j \in \mathcal{M}. \quad (9)$$

With the performance model in Section II, the interactions among p_j, q_j, v_j, and n_j at equilibrium can therefore be jointly defined by Equations (1), (7), (8), and (9).

IV. Revenue Maximization for 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 the prior knowledge of users' responses (i.e., service class selections) to the 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 configuration parameters b_j, c_j, and r_j. We also consider n_0 and v_0 as provider-specified, which characterize the provider's preference in accepting users. (For example, a provider that wishes to accommodate every user will take $p_0 = 0$ and $v_0 = 0$ with all users being accepted.) 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 derived in Sections II and III, we formulate the revenue maximization problem as

$$\underset{\mathbf{p},\mathbf{q},\mathbf{n},\mathbf{v}}{\text{maximize}} \quad \sum_{j=1}^{M} p_j n_j, \tag{10}$$

$$\text{subject to} \quad \text{constraints (1), (7), and (9)},$$

$$q_j \leq q_{j+1},\ \forall j \in \mathcal{M} \setminus \{M\}, \tag{11}$$

$$n_j \in \mathbb{Z}^+,\ \forall j \in \mathcal{M}. \tag{12}$$

In the objective function (10), the provider's total revenue is the summation over the revenue $p_j n_j$ gained by each service class j. Constraints (1), (7), and (9) jointly define the relationship among the decision variables at equilibrium. Since vector $\mathbf{q}$ is sorted in a non-decreasing order at equilibrium, without loss of generality, we configure the service classes as $b_j \leq b_{j+1}$, $c_j \leq c_{j+1}$, $r_j \leq r_{j+1}$, $j \in \mathcal{M} \setminus \{M\}$. Therefore, it is natural to expect that service classes with richer resources will offer higher QoS levels, as indicated in Constraint (11).

The revenue maximization problem is a non-linear integer program, a hard problem in general. Due to the limited space, we propose to solve the problem by general-purpose methods [18] in this paper, and leave the design of a customized approximation algorithm as our future work. General-purpose methods are likely sufficient for determining the optimal burstable instance configurations as these configurations usually do not change much over time (*cf.* Section V-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 [4]; released in 2017, these traces are the latest characterization of VM resource utilization in public clouds.

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	9	2,592	300		0.1	
4	14	4,032	400		0.15	
5	19	5,472	500		0.2	

A. Framework Validation

1) Validating the performance model: The Microsoft Azure traces record CPU utilization of VMs at a time granularity of 5 minutes. Therefore, the duration of a time slot in our simulations is also set to be 5 minutes, and a token refers to 1% of the full capacity of a vCPU for 5 minutes. We examine five different service class configurations listed in Table II. We set the token bucket sizes b_j as the number of tokens earned in 24 hours, and the number of initial tokens for a service class as $1/6$ of the token bucket size for a smooth bootstrap. Since the average resource volume received per instance is no larger than 20% according to Table II, we exclude the VMs with average CPU utilizations higher than 20%, 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 relatively high CPU requirements.)

We sort the instance records in chronological order, and randomly select 100 of the first 2,600 records[4] as samples to estimate δ, the probability that a user initiates a burst request, 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 2,500 instance records are used as *testing* data in the simulations of this sub-section. The δ value and the θ distribution are respectively obtained by simply counting the number of times that users have burst requests to send, and the frequencies of appearance for different θ values in the 100 sample instance records. Our obtained δ value is 0.9970.

We simulate a total time span of 5 days, and play back the workloads in the traces. Figure 2a shows the simulated QoS as we vary the number of users n_j in each of Table II's service classes from 1 to 100. For each service class j with a given number of users n_j, the figure shows the average QoS over 20 runs with n_j instance records randomly drawn from the 2,500 testing records in each run. We also plot two representative comparisons between our analytical (from Section II) and simulated QoS curves in Figures 2b and 2c for service classes

[4]All of these 2,600 instances start in the first time slot of the time horizon, and have durations longer than 5 days.

$j = 1$ and $j = 5$, respectively. Qualitatively, it can be observed that our analytical curves are close to the simulated curves. The average error ratios of our analytical curves for service classes $j = 1$ and $j = 5$ are 1.91% and 2.15%, respectively, which are relatively small. Thus, our analytical performance model can both qualitatively and quantitatively reflect the actual QoS.

2) Validating the equilibrium analysis and revenue maximization: Consider that users' QoS valuations u_i follow a uniform distribution within $(0, 1]$. The CDF at v_j is thus

$$f(v_j) = v_j, \ j = 0, 1, \ldots, M. \tag{13}$$

Substituting Equation (13) into (7), (8), and (9), we obtain the analytical equilibrium representation.

To evaluate our revenue maximization scheme, we compare it with two benchmarks, the *uniform* and the *random* schemes, with different numbers of users N and the service class configurations in Table II. Both benchmarks 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 $n_0/N = 0.05$. Among the remaining $N - n_0$ users, the *uniform* scheme sets n_j equally while the *random* scheme selects n_j, $j \in \mathcal{M} \setminus \{5\}$ from a normal distribution with mean $(N - n_0)/5$ and standard deviation $(N - n_0)/15$, and n_5 is calculated by Equation (7) afterwards. The service class prices are then determined by Equations (1) and (9) to ensure that they constitute a Nash equilibrium. The other simulation settings stay unchanged from those presented in Section V-A1. By varying the total number of users N, the corresponding revenues under the three schemes are shown in Figure 3, where our *optimal* scheme always derives the maximum revenue, around 50% higher than the revenues generated by the other two benchmarks.

TABLE III: Service-class-wise results for the $N = 300$ case in Figure 3. The result that generates the median revenue over 25 runs for the *random* scheme is reported.

j		1	2	3	4	5
Optimal	p_j	0.0055	0.3551	0.4636	0.5034	0.5072
	$\overline{q}_j$	0.1055	0.7435	0.8481	0.9591	0.9656
	$\overline{u}_i\overline{q}_j$	0.0319	0.4436	0.5884	0.7741	0.8907
	n_j	146	26	33	36	44
	v_j	0.5367	0.5900	0.7333	0.8533	1.0000
Random	p_j	0.0215	0.0221	0.0587	0.1967	0.4092
	$\overline{q}_j$	0.3898	0.3922	0.4823	0.6763	0.8701
	$\overline{u}_i\overline{q}_j$	0.0478	0.1224	0.2598	0.5062	0.7997
	n_j	40	68	69	56	52
	v_j	0.1833	0.4100	0.6400	0.8267	1.0000
Uniform	p_j	0.0149	0.0672	0.1168	0.1741	0.3320
	$\overline{q}_j$	0.2757	0.4604	0.5877	0.6528	0.8134
	$\overline{u}_i\overline{q}_j$	0.0426	0.1614	0.3160	0.4775	0.7411
	n_j	57	57	57	57	57
	v_j	0.2400	0.4300	0.6200	0.8100	1.0000

We next investigate the reasons why our *optimal* scheme outperforms the others. For each service class j, we elaborate the price (p_j), simulated QoS on average ($\overline{q}_j$), simulated user utility on average ($\overline{u}_i\overline{q}_j$), number of admitted users (n_j), and the corresponding QoS valuation boundary points (v_j) for the three schemes under the $N = 300$ case in Table III. It can be observed that our *optimal* scheme attaches more importance to improving the QoS offered by service classes 2 to 5 by restricting them to few users. These users also have higher QoS valuations u_i according to Lemma 2, resulting in higher utilities ($u_i q_j$) achieved. They are thus willing to pay higher prices, leading to an increase of the provider's revenue.

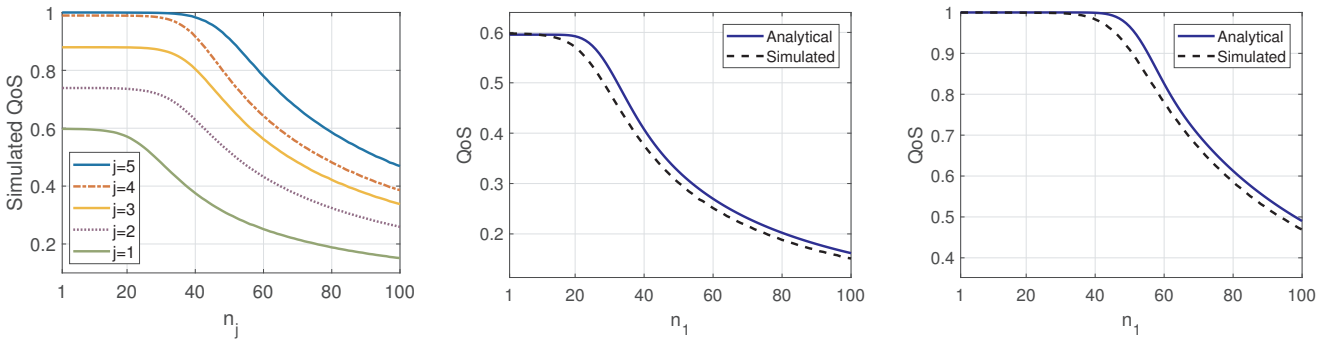
3) Impact of n_0 on the revenue: From the provider's perspective, n_0/N can be interpreted as the acceptance rate of users. A smaller n_0/N means more users whose QoS valuations satisfy $u_i \in [v_0, \gamma]$ will be admitted by service classes at equilibrium. To gain insights into how n_0 influences the revenue, we vary n_0 with N fixed, and report the corresponding *optimal* revenue in Figure 4. When n_0 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 each service class. According to Equation (5), 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 is too high, the number of admitted users becomes 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.

B. 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. We use the Microsoft Azure traces as the VM workloads in the cloud. 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 5-day period from day 3 to day 7.

To accommodate the large number of simultaneously running VMs in the traces, more resources are needed. However, simply increasing the resource capacity c_j of a service class may make the service class not implementable on one single physical server, leading to VM migration issues. Therefore, we duplicate each service class j in Table II by 450 times, and refer to such a duplicated service class as a type-j service class. In this case, we have $2,250$ service classes from five types in total. The other parameters stay the same as those in Section V-A2. We set $N = 135,000$, corresponding to the peak number of VMs in the system over time. We then partition the service classes into 450 groups, with five different types of service classes and $N/450$ users in each group. A group can be practically implemented as a server with 18 ($= a^{grntd} \cdot N/450 + \sum_{j=1}^{5} c_j$, 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 under the *optimal*, *uniform*, and *random* schemes within a group at equilibrium, the same to that in Section V-A2.

The cloud assigns VMs to service classes as follows to tackle the dynamic creation and termination of VMs. When a new VM i needs to be created, we first examine its u_i to determine



(a) All simulated QoS curves. (b) QoS curves for service class $j = 1$. (c) QoS curves for service class $j = 5$.

Fig. 2: (a) presents the simulated QoS curves for all five service classes. (b) and (c) compare the analytical and simulated QoS curves for two representative service classes, whose parameters are listed in Table II. The error ratios of our analytical curves are 1.91% and 2.15% for (b) and (c), respectively. In summary, our analytical performance model can reflect the actual QoS.

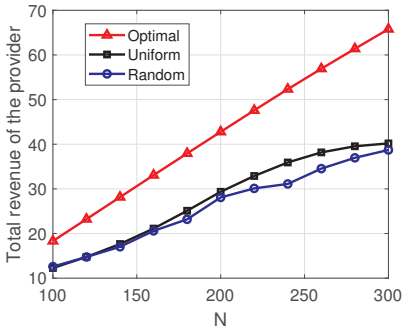


Fig. 3: Total revenues under different schemes for Section V-A2.

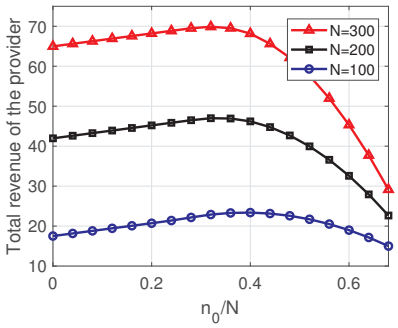


Fig. 4: Optimal total revenues under different n_0/N rates for Section V-A3.

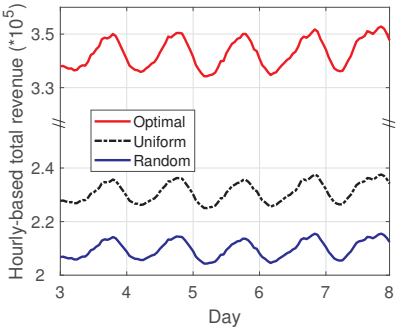


Fig. 5: Hourly revenue over 5 days for the case study in Section V-B.

which service class type it should belong to. The VM is then assigned to the service class of this type that has the minimum number of active VMs, i.e., VMs that are currently running. As the QoS and prices are designed with regard to the peak demand, the number of active 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 scheme during off-peak periods. When an existing VM terminates, we remove it from its belonged 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 $12p_j$ in total. The hourly-based revenues in the time horizon under three different schemes are presented in Figure 5. Our *optimal* scheme is shown to yield the best revenue, which is around 50% higher than the revenues generated by the benchmarks.

VI. Related Work

Existing works on burstable instances mainly report empirical measurement results and discuss use cases. A use case for backup servers is presented in [10]. Through extensive measurements, Leitner *et al.* [19] have verified that the CPU credit mechanism works as advertised by cloud providers, and Wang *et al.* [15] further point out that this mechanism essentially follows a token bucket model. This finding has motivated us to model burstable instances and analytically study the performance. To the best of our knowledge, our work is the first to study burstable instances from a theoretical perspective. Yan *et al.* [20] explore the initial CPU credits assigned to burstable instances for smooth bootstraps. They study how to shape the CPU utilizations of applications to make use of the initial CPU credits, while our work focuses on the stationary behaviors of burstable instances after bootstrapping.

Similar techniques to those employed in our work, including the token bucket model and optimal pricing, have been used in the literature to address different problems. For instance, token bucket models have been extensively adopted to regulate data traffic [21], [22]. Other works have priced service classes with differentiated QoS levels in data networks [23], [24]. However, due to the different system dynamics and characteristics, these results cannot be directly applied to our burstable instance scenario. In the same vein, the distinct features of burstable instances compared to traditional cloud instances with static resource provisioning prevent existing models on cloud pricing (e.g., [25]) from being applied to our scenario.

Some early works have proposed alternative resource provisioning ideas to tackle bursty workloads in clouds. Wang *et al.* [26] 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 [27]. The brokerage strategy follows a different business model and

system characteristics compared to ours. The model we study stems from current practices in the industry. Another stream of works has investigated, from the applications' perspective, how burst requests should be made via proactive prediction [28], [29] or online algorithmic decision processes [30], [31], while our work focuses on how burst requests already made by users can be accommodated by a cloud provider.

VII. Conclusion

This paper introduces a framework to analytically model the performance of burstable instances given the 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. We show that our performance model can estimate the QoS received by burstable instances with a small average error ratio, and our revenue maximization scheme can drastically increase the provider's revenue.

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 this work, we can investigate a customized approximation algorithm for the revenue maximization problem in our framework. Such an algorithm may allow our framework to adapt to changes in users' resource demand patterns over time.

Acknowledgment

We thank Hedyeh Beyhaghi, Ting-Jung Chang, Zhe Huang, Tri Nguyen, Liang Zheng, and anonymous reviewers for their feedback. This material is based on research sponsored 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.

References

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

[2] 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.

[3] 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.

[4] 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. Princ.*, 2017, pp. 153–167.

[5] 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. Autom. Comput.*, 2010, pp. 11–20.

[6] "Amazon EC2: T2 standard." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-std.html

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

[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] "Amazon EC2: CPU credits and baseline performance." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-credits-baseline-concepts.html

[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. Eur. 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] K. Ramamritham, "Real-time databases," *Distrib. Parallel Databases*, vol. 1, no. 2, pp. 199–226, 1993.

[13] S. A. Golder, D. M. Wilkinson, and B. A. Huberman, "Rhythms of social interaction: Messaging within a massive online network," in *Communities Technol.*, 2007, pp. 41–66.

[14] 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.

[15] C. Wang, B. Urgaonkar, N. Nasiriani, and G. Kesidis, "Using burstable instances in the public cloud: Why, when and how?" *Proc. ACM Measurement Anal. Comput. Syst.*, vol. 1, no. 1, pp. 11:1–11:28, 2017.

[16] 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." [Online]. Available: http://c2e.ece.ust.hk/papers/infocom19-tr.pdf

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

[18] D. Li and X. Sun, *Nonlinear integer programming*. Springer Science & Business Media, 2006, vol. 84.

[19] 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.

[20] 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.

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

[22] 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. Qual. Service*, 2015, pp. 105–114.

[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] A. Odlyzko, "Paris metro pricing for the internet," in *Proc. ACM Conf. Electron. Commerce*, 1999, pp. 140–147.

[25] 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.

[26] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 400–409.

[27] 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 Perform. Eval. Rev.*, vol. 44, no. 1, pp. 235–248, 2016.

[28] 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. Perf. Comput. Commun. Conf.*, 2011, pp. 1–8.

[29] D. J. Dubois and G. Casale, "Performance prediction for burstable cloud resources," in *Proc. EAI Int. Conf. Perform. Eval. Methodologies Tools*, 2017, pp. 217–218.

[30] 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.

[31] N. Morris, C. Stewart, L. Chen, R. Birke, and J. Kelley, "Model-driven computational sprinting," in *Proc. Eur. Conf. Comput. Syst.*, 2018, pp. 38:1–38:13.