On the Viability of a Cloud Virtual Service Provider

Liang Zheng* Carlee Joe-Wong* Christopher G. Brinton*
Chee Wei Tan† Sangtae Ha§ Mung Chiang*
*Princeton University †City University of Hong Kong §University of Colorado
{liangz, cjoe, cbrinton, chiangm}@princeton.edu cheewtan@cityu.edu.hk sangtae.ha@colorado.edu

ABSTRACT

Cloud service providers (CSPs) often face highly dynamic user demands for their resources, which can make it difficult for them to maintain consistent quality-of-service. Some CSPs try to stabilize user demands by offering sustained-use discounts to jobs that consume more instance-hours per month. These discounts present an opportunity for users to pool their usage together into a single "job." In this paper, we examine the viability of a middleman, the cloud virtual service provider (CVSP), that rents cloud resources from a CSP and then resells them to users. We show that the CVSP's business model is only viable if the average job runtimes and thresholds for sustained-use discounts are sufficiently small; otherwise, the CVSP cannot simultaneously maintain low job waiting times while qualifying for a sustained-use discount. We quantify these viability conditions by modeling the CVSP's job scheduling and then use this model to derive users' utility-maximizing demands and the CVSP's profit-maximizing price, as well as the optimal number of instances that the CVSP should rent from the CSP. We verify our results on a one-month trace from Google's production compute cluster, through which we first validate our assumptions on the job arrival and runtime distributions, and then show that the CVSP is viable under these workload traces. Indeed, the CVSP can earn a positive profit without significantly impacting the CSP's revenue, indicating that the CSP and CVSP can coexist in the cloud market.

1. INTRODUCTION

Cloud computing is forecasted to grow to a market size of $112 billion in 2018 [11], in large part due to its Infrastructure-as-a-Service (IaaS) offerings. IaaS attracts customers from individuals and organizations of all sizes, who can benefit from pay-as-you-go pricing that allows users to rent computing resources by the hour, eliminating setup and maintenance costs. Yet as more cloud service providers (CSPs) offer these services, the IaaS market has become increasingly competitive, with CSPs competing to offer ever-lower prices to users [1, 8, 30]. Compounding this difficulty, the number and variety of tasks that users submit to the cloud create a highly dynamic environment with many jobs that require different amounts of resources arriving and departing at different times. CSPs do take advantage of this job heterogeneity to optimize their capacity provisioning, but their pay-as-you-go pricing means that they cannot completely prevent localized instances of high demand and high user waiting times [21, 26, 27, 38]. Thus, many providers are turning to different pricing plans to match their IaaS prices to real-time user demands.

SIGMETRICS '16, June 14-18, 2016, Antibes Juan-Les-Pins, France

DOI: http://dx.doi.org/10.1145/2896377.2901452

1.1 Existing Cloud Pricing

IaaS resources are generally virtualized in units of instances. Each instance consists of a remote virtual machine (VM) with specified amounts of CPU, memory, storage, and other attributes. Users submit jobs by requesting the use of their desired types of instances. Each job consists of one or multiple parallel or dependent tasks, with each task run on a single instance and charged collectively with other tasks in the job. Users pay for the execution of their jobs per instance per hour. Most CSPs offer three types of instance pricing: usage-based, auction-based, and volume-discount pricing, as summarized in Table 1.

Usage-based pricing, also known as pay-as-you-go, is the simplest pricing scheme, charging users at a fixed unit (per-instance, per-hour) price [30, 40]. Auction-based pricing, such as Amazon's Elastic Compute Cloud (EC2) spot pricing [4] and Google's preemptible VMs [15], introduces more freedom to respond to real-time market demand: users bid for instances and the CSP sets dynamic thresholds for successful bids. While several works have shown that user cost can be reduced by such bidding, jobs run under this scheme may be interrupted prior to completion [22, 24, 39].

Volume-discount pricing offers another way for CSPs to stabilize user demand, without auction pricing's potential for interruptions. Under this scheme, users are charged a lower unit price if their usage exceeds a given threshold, *e.g.,* 25% of a one-month billing cycle for Google Cloud Platform [14]. Amazon EC2 allows users to rent reserved instances for 1 or 3 years at a reduced rate [3]. In our work, we follow Google's terminology and use *sustained-use discount* to refer to the discount afforded to longer-lasting jobs. We focus on this type of pricing in the paper.

1.2 The Cloud Virtual Service Provider

Sustained-use discounts appear to only benefit users with longer-lasting jobs. However, if users with shorter jobs could

Table 1: Different types of cloud pricing.

Scheme	Job Runtime	Unit Price	Typical Job Types
Usage-based	Short-term	Highest	Scientific computing, backend batch processing
Auction-based	Interruptible; no guaranteed deadline	Lowest	Word counting, multimedia processing
Volume-discount	Long-term	Discounted	Real-time web service, remote monitoring

pool their jobs together, they could jointly create a *virtualized* longer "job" to take advantage of the sustained-use discount.[1] This creates an opening for cloud virtual service providers (CVSPs) to rent long-term instances from CSPs offering these discounts. The CVSP can then attract users with shorter jobs by selling them this rented capacity at a small markup: users still pay lower per-instance, per-hour prices than they would by buying directly from the CSP, and the CVSP can also make a profit. Figure 1 illustrates this pooling scheme: individual users consume relatively little resources, but when pooled together they take up most of the VM time. In the remainder of the paper, we use CSP to denote the cloud service provider from which the CVSP rents instances to resell to users.

This type of user pooling to resell resources has been practiced in a variety of other contexts, *e.g.,* mobile virtual network operators (MVNOs) in the U.S. often offer cheaper mobile data plans to their users than the four major network operators [23]. Many cloud storage companies, like Dropbox, sell a version of virtualized cloud resources in which they store user data on rented VMs and add services like file sharing on top of the storage resources [12]. No work has yet studied virtual cloud services in which VMs are only resold to users without significant additional services.

While their business model seems generally reasonable, more careful consideration reveals some uncertainty as to whether CVSPs can viably support their users' demands. In particular, CVSPs need to balance offering lower prices to users with providing acceptable quality-of-service (QoS) to them. On the one hand, if the CVSPs do not attract enough users to their platform, they will not achieve the high usage volumes necessary to receive the CSP's sustained-use discounts, and must charge users higher prices. On the other hand, attracting too many users could cause congestion, forcing users to wait long amounts of time before their jobs complete. Indeed, in a worst-case scenario, users' waiting times may grow to infinity as congestion builds up at the CVSP. Users may then decide to move their jobs to the CSP, even if it is more expensive. It is therefore unclear *whether virtual CSPs can viably exist in the IaaS market.* We establish conditions for CVSPs' viability in this paper.

1.3 Research Challenges and Contributions

The viability of a CVSP's business model depends on users' demands for its services: while the CVSP must attract enough users to qualify for the sustained-use discounts, it must also keep their demands low enough to provide acceptable QoS in order to compete with the CSP. We take QoS to be determined by users' waiting times in this paper

[1]If a CVSP attracts too many users, the CSP may refuse to rent VMs to the CVSP to avoid cannibalizing its business. Full consideration of the CSP's actions is beyond the scope of this paper, but we show numerically that the CVSP does not completely take over the CSP's business, allowing both to exist in the market (Section 5).

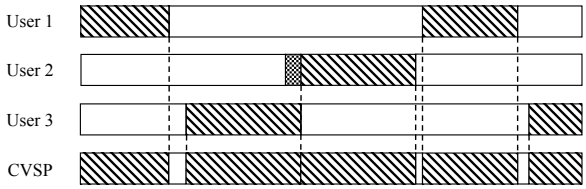


Figure 1: A simple one-instance example of a CVSP that pools usage from three users, reselling them VMs from CSPs. Shaded regions indicate when jobs are running for each user, while the grid-shaded region indicates that user 2 has to wait until user 3 finishes. Blank regions indicate idle time.

and define a CVSP's *viability* as its ability to qualify for sustained-use discounts while providing finite waiting times for some fixed, nonzero level of user demand.

In reality, users' demands are not fixed; they are influenced by the CVSP's decisions. CVSPs that charge users very low prices, for instance, will attract many users to their platform. However, these high demand levels may force users to wait long amounts of time for their jobs to run. Even if the CVSP is viable at a given demand level, if users' waiting times are finite but long, then they may not submit their future jobs to the CVSP. Yet this effect also depends on the number of VMs rented by the CVSP: if the CVSP has rented enough VMs, then user jobs could be run simultaneously, possibly compensating for the congestion induced by lower CVSP prices. Complicating matters is the sustained-use discount offered by the CSP that may increase with the CVSP's usage time on an instance, further increasing the CVSP's profit at high user demands. Thus, it is unclear *what prices the CVSP should charge its users* or *how many VM instances it should rent from the CSP.*

In order to model these CVSP decisions, we first develop a model for users' received QoS, given their demand levels, and use it to assess the CVSP's viability. We then evaluate users' strategically optimal demands and the CVSP's corresponding optimal economic strategy. We finally verify our results with evaluations on a one-month dataset of cloud workload traces. After a brief review of related work in Section 2, this paper establishes the following:

CVSPs' operational viability (Section 3): To determine the CVSP's viability, we first evaluate users' received QoS, given their demand levels. As such, we develop a model to characterize the system under Poisson-distributed user arrivals [29, 36, 37] and Pareto-distributed job running times [10, 25]. We then find the expected user waiting time and conditions under which it remains bounded. On relating these conditions to the expected system idle time, we show that the CVSP faces a tradeoff between lowering the user waiting time and lowering the system idle time to qualify for sustained-use discounts. We find that if users' average

job runtimes are sufficiently small, then the CVSP can ensure finite waiting times and qualify for the discounts simultaneously: though shorter jobs consume fewer resources, sufficiently many short jobs will both consume enough resources to qualify for sustained-use discounts and allow for shorter user waiting times. However, if the threshold for the sustained-use discounts is too high, then the CVSP is not viable for any job runtimes: users' waiting times will grow to infinity at usage levels compatible with these discounts.

The optimal CVSP strategy (Section 4): We use our system characterization to formulate the CVSP's optimal (profit-maximizing) strategy, *i.e.,* its optimal prices and number of VMs. We first introduce a model of user demand, in which we suppose that users choose the frequency of job submissions and the number of jobs that they submit to the CVSP depending on the CVSP's QoS and price provided to users. We therefore observe a dependence among different users' actions: each user's demand affects the average waiting time in the system, which in turn affects other users' optimal demands. We find the equilibrium job arrival rate given this dependence, and then use the viability conditions derived in Section 3 to find the CVSP's profit-maximizing price and number of VMs to rent.

Experiments on a real-world data trace (Section 5): We finally simulate our model on a real-world user trace that contains the scheduling events of more than 600,000 jobs and 25 million tasks over one month. We first verify that users' job arrival rates follow a Poisson distribution and that the job runtimes follow a Pareto distribution. We then simulate the market dynamics, using these to verify that by choosing the right price to charge users and number of VMs to rent from the CSP, the CVSP can (i) obtain a sustained-use discount while maintaining a tolerable waiting time for users; and (ii) make a positive profit. Moreover, we show that the CVSP's market share is limited, giving the CSP more incentive to allow the CVSP to exist: the CVSP rents only a portion of the CSP's total capacity, stabilizing demand for this portion of the CSP's capacity and leaving the rest for the CSP to market directly to users.

2. RELATED WORK

Cloud and datacenter workloads have been extensively studied. User requests are often modeled as Poisson processes for analytic convenience [2, 35], though a number of early studies showed that the workload interarrival times experienced by traditional datacenters were not exponentially distributed [17]. However, as datacenter services become more popular and users become more experienced in scaling and managing their workloads, these traffic dynamics are changing. Recent measurements on different traces have suggested that job arrivals are in fact Poisson distributed [29, 36, 37], consistent with our own findings in this work. Furthermore, other works have used the Pareto distribution to model job service times [10, 25], as we do here, while still others have characterized cloud workloads without leveraging user behavior [21, 31].

Other works have considered cloud resource provisioning and dynamic scheduling from a purely operational perspective [2, 5, 6, 16, 26, 38]. Power usage is a particularly important concern for CSPs, *e.g.,* the authors of [26, 38] used the heterogeneity of VM resources and workloads to minimize power consumption and scheduling delay, while [5, 6] considered the tradeoffs between power consumption and scheduling. Furthermore, authors in [13] derived equilibrium algorithms for cloud resource scheduling. While the CVSP does need to schedule user jobs on its rented instances, it does not need to provision resources or consider energy costs, leaving these to the CSP.

Some works have treated cloud pricing as a game between CSPs and the users to reach an equilibrium [7, 9, 33]. As an alternative, others have used auctions for distributed cloud resource allocation [18, 32, 39]. Given that user demands are crucial to datacenter efficiency, the authors in [19] designed a prediction-based pricing scheme for demand response in data centers. We instead propose using the CVSP as a middleman to mitigate uncertainty in user demand, thus reducing user cost and increasing operational efficiency for CSPs.

Table 2: Key terms and symbols.

Symbol	Definition
L	Number of users who subscribe to the CVSP.
M	Number of instances that the CVSP rents from the CSP.
N	Expected number of tasks within one job.
λ_l	Poisson distribution parameter for the lth user's arrival rate.
Λ	Poisson distribution parameter for the system arrival rate, where $\Lambda = \sum_{l=1}^{L} \lambda_l$.
$\tilde{\Lambda}$	Instance arrival rate, where $\tilde{\Lambda} = N\Lambda/M$.
t	Time duration of two adjacent job arrivals.
$\tau, \underline{\tau}$	Job runtime and the minimum runtime.
α	Pareto distribution parameter for job runtime.
φ	Expected waiting time.
T	A month's time.
θ	The expected idle-to-runtime ratio of an instance's usage
ϵ	Fraction of an instance's usage of the total time in a billing cycle.
p	Unit price that users pay the CVSP.
π	Full unit price without sustained usage that the CSPs charge.
ω	Fraction of full price that the CVSP pays.

3. CVSP VIABILITY

In this section, we consider the CVSP's ability to balance user QoS with qualifying for sustained-use discounts. To do so, we analyze a simple job scheduler for the CVSP that allows us to find users' expected job waiting times and the expected system idle time. We suppose that users' jobs are all run on the same type of VM instances and that they have equal priorities: each job request is fulfilled as soon as enough CVSP instances are available.[2] We first formalize the user arrival distribution (Section 3.1), and then derive the job waiting time and the system idle time, using these to establish CVSP viability in Section 3.2.

3.1 System Model

We consider a system of L users, each of whom is expected

[2]While the scheduler does not consider all nuances of user jobs, *e.g.,* dependencies between job tasks, we expect that its qualitative results will hold in practice, and we numerically validate our assumptions in Section 5.

to submit jobs consisting of N tasks at a time.[3] Each task is required to run on a separate instance, *i.e.,* they cannot simultaneously share resources on the same instance. We suppose that the CVSP has rented M ($M \geq N$) instances to allocate to its users. Without loss of generality, we assume that the lth user's job arrival rate follows a Poisson distribution with mean λ_l, $l = 1, 2, \ldots, L$ [29, 36, 37]. Thus, the arrival rate from the point of view of the system (our focus in this section) is denoted by $\Lambda = \sum_{l=1}^{L} \lambda_l$, and the time t between two job arrivals is exponentially distributed. As a heavy-tailed distribution is often used to model job runtimes [10, 25], we assume that the time τ for which each job will run in the system follows a Pareto distribution with parameter $\alpha > 1$ and a minimum runtime $\underline{\tau}$.[4] We suppose that each task within a given job must run for the same amount of time, which has the expected value $\frac{\alpha\underline{\tau}}{\alpha-1}$. All times in this paper are assumed to be given in hours. We summarize our notation in Table 2.

We assume that users request resources from the CVSP independently over a continuous time interval. After receiving a job request of N tasks from a user, the CVSP randomly allocates the tasks to N instances. Since we do not distinguish between different instances, each instance has the same probability of being available at any given time, and thus of being assigned a task. The probability that an instance will be chosen for a given job request is then $1 - \binom{M-1}{N}/\binom{M}{N} = \frac{N}{M}$, where $\binom{M}{N}$ is the binomial coefficient representing the number of combinations of selecting N items from a set of M items. Now, it can be established that the task arrival rate for each instance also has a Poisson distribution:

LEMMA 1. *If (i) job arrivals follow a Poisson distribution with parameter Λ, (ii) each submitted job requires N tasks, and (iii) the CVSP can run the jobs on M instances, then the task arrival rate for each of the instances follows a Poisson distribution with parameter $\frac{N}{M}\Lambda$.*

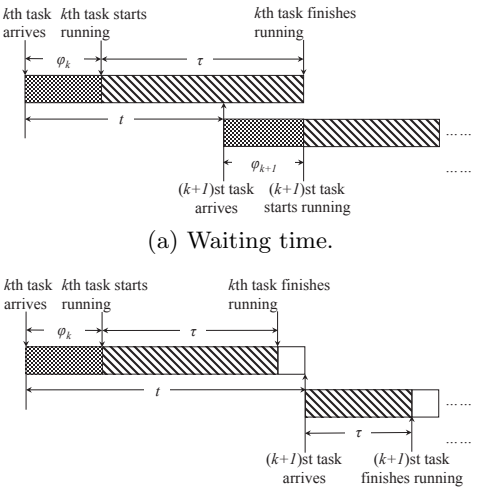
PROOF. Suppose that for a given instance, t is the time interval between the starting times of two adjacent requests. We can derive the probability density function of t:

$$f(t) = \sum_{j=1}^{\infty} \left(\frac{N}{M}\right)\left(1 - \frac{N}{M}\right)^{j-1} \frac{\Lambda^j t^{j-1} e^{-\Lambda t}}{(j-1)!}$$
$$= \frac{N}{M}\Lambda e^{-\Lambda t} \sum_{j=1}^{\infty} \frac{\left((1-\frac{N}{M})t\right)^{j-1}}{(j-1)!} = \frac{N}{M}\Lambda e^{-\frac{N}{M}\Lambda t}.$$

To understand the above derivations, we note that $\frac{\Lambda^j t^{j-1} e^{-\Lambda t}}{(j-1)!}$ is the probability of the jth Poisson arrival in time t according to the Erlang distribution, $\left(1 - \frac{N}{M}\right)^{j-1}$ is the probability that the first $j-1$ users' jobs are all assigned to the other instances, and one of the jth user's jobs is assigned to this instance with probability $\frac{N}{M}$. Since $f(t) = \frac{N}{M}\Lambda e^{-\frac{N}{M}\Lambda t}$ is the probability density function of an exponential distribution with parameter $\frac{N}{M}\Lambda$, the job arrival rate for this instance follows the Poisson distribution with parameter $\frac{N}{M}\Lambda$. □

Henceforth, we refer to Λ and $\frac{N}{M}\Lambda$ as the *system arrival rate* and the *instance arrival rate*, respectively. For ease of notation, $\tilde{\Lambda} = \frac{N}{M}\Lambda$ will denote the instance arrival rate.

Although the result in Lemma 1 is not qualitatively surprising, it allows us to quantify the expected time between two task arrivals at an instance as $\tilde{\Lambda}^{-1} = \frac{M}{N\Lambda}$. Thus, if the next task is expected to arrive before the current task finishes running, *i.e.,* $\tilde{\Lambda}^{-1} < \frac{\alpha\underline{\tau}}{\alpha-1}$ (recall $\frac{\alpha\underline{\tau}}{\alpha-1}$ is the expected task runtime), we would expect to observe larger waiting times that could diverge to infinity. Conversely, if the expected task runtime is shorter than the expected time between adjacent task arrivals, *i.e.,* $\tilde{\Lambda}^{-1} \geq \frac{\alpha\underline{\tau}}{\alpha-1}$, the system may experience idle time between jobs. We formalize this intuition in the next section.



(a) Waiting time.

(b) Idle time.

Figure 2: Waiting time and idle time between two consecutive task arrivals. A new task must wait if it arrives before all previous tasks have finished running. On the other hand, there will be idle time for the system if the next task has not arrived by the time the previous task has finished running.

3.2 Balancing QoS with System Idle Time

We now evaluate the CVSP's viability using the above system model by first finding the conditions for finite job waiting times and then the conditions under which the CVSP's system idle time is low enough to qualify for sustained-use discounts.

3.2.1 User Waiting Time

We assume that each instance runs its assigned tasks in a first-come-first-served manner: a new arrival to an instance will only begin to run as soon as all of the tasks that arrived before it have completed. We let φ_k denote the waiting time of the kth arrival, as illustrated in Figure 2: the kth task first waits for time φ_k and then spends time τ to run. Two possibilities can then happen to the $(k+1)$st task: either it arrives before the kth task has finished (Figure 2(a); $t < \varphi_k + \tau$) and has to wait for time $\varphi_{k+1} = \varphi_k + \tau - t$ before it is launched, or it arrives after the kth task finishes (Figure 2(b); $t \geq \varphi_k + \tau$) and does not wait ($\varphi_{k+1} = 0$).

Taking these two possibilities into consideration, the expected waiting time of the next task arrival can be written

[3] Our results remain the same if N instead represents the expected number of tasks per job; we assume in this section that each job has N tasks for simplicity.

[4] On Google's Cloud Platform, all jobs run for a minimum of 10 minutes, or $\underline{\tau} = 1/6$ of an hour.

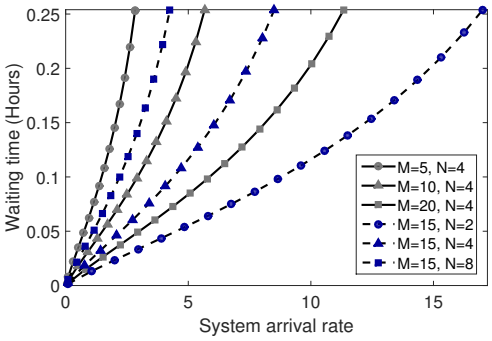


Figure 3: An illustration that the expected waiting time increases with the system arrival rate Λ.

in terms of the waiting time of the previous task: $\varphi_{k+1} = \int_{\underline{\tau}}^{\infty} \left(\int_0^{\varphi_k+\tau} (\varphi_k + \tau - t)\tilde{\Lambda} e^{-\tilde{\Lambda}t} \mathrm{d}t \right) \frac{\alpha \underline{\tau}^{\alpha}}{\tau^{\alpha+1}} \mathrm{d}\tau$, where we have abused on notation and use φ_{k+1} to refer to the expected waiting time of the $(k+1)$st task instead of the actual waiting time. This expression can be further simplified to

$$\varphi_{k+1} = \varphi_k - \frac{1}{\tilde{\Lambda}} + \frac{\alpha}{\alpha - 1}\underline{\tau} + \frac{\alpha}{\tilde{\Lambda}} \left(\tilde{\Lambda}\underline{\tau}\right)^{\alpha} e^{-\tilde{\Lambda}\varphi_k} \Gamma\left(-\alpha, \tilde{\Lambda}\underline{\tau}\right), \quad (1)$$

where $\Gamma(s, z) \equiv \int_z^{\infty} x^{s-1} e^{-x} \mathrm{d}x$ is the upper incomplete gamma function.

Note that (1) is a fixed-point function for the expected task waiting times, and, in particular, is a recursion $\varphi_{k+1} = g(\varphi_k)$ for the implied function g. If the CVSP does not rent enough instances, *i.e.,* if M is too small, it is possible that φ_{k+1} will diverge to infinity. In such cases, the CVSP would not be viable, since users would eventually have to wait forever. To ensure finite waiting times, we derive a lower bound on M for which the waiting time converges to a fixed point:

PROPOSITION 1. *If the number of instances rented by the CVSP satisfies*

$$M \geq \left\lceil \frac{\alpha N}{\alpha - 1} \Lambda \underline{\tau} \right\rceil, \quad (2)$$

then asymptotically as $k \to \infty$, all of the tasks have a finite expected waiting time

$$\varphi = \frac{1}{\tilde{\Lambda}} \log \left(\frac{\alpha \left(\tilde{\Lambda}\underline{\tau}\right)^{\alpha} \Gamma(-\alpha, \tilde{\Lambda}\underline{\tau})}{1 - \frac{\alpha \underline{\tau}}{\alpha - 1}\tilde{\Lambda}} \right), \quad (3)$$

where log *denotes the natural logarithm.*

PROOF. We show that the fixed-point iteration (1) converges if and only if (2) holds. For ease of notation, we first rewrite (1) by replacing $a = \frac{1}{\tilde{\Lambda}} - \frac{\alpha}{\alpha-1}\underline{\tau}$ and $b = \frac{\alpha}{\tilde{\Lambda}} \left(\tilde{\Lambda}\underline{\tau}\right)^{\alpha} \Gamma\left(-\alpha, \tilde{\Lambda}\underline{\tau}\right)$ and obtain:

$$g(\varphi_k) = \varphi_k - a + b e^{-\tilde{\Lambda}\varphi_k}.$$

To show that (2) is a necessary condition, we note that (2) is equivalent to $a \geq 0$. If $a < 0$, then since $b > 0$, we have $g(\varphi_k) \geq \varphi_k - a = \varphi_0 - ka \to \infty$ as $k \to \infty$, so the iteration cannot converge.

To show that (2) is a sufficient condition, we use Banach's fixed-point theorem. We first show that $g(\varphi_k)$ maps $[0, R]$ to itself for some $R > 0$, and then that $g'(\varphi_k) \in (-1, 1)$ for all φ, *i.e.,* that g is a contraction mapping on $[0, R]$. Since $\varphi_0 = 0$, i.e., the first waiting time is 0 (since there are no preceding tasks), this is sufficient to show convergence.

We first show that if $\varphi_k \in [0, R]$, then $g(\varphi_k) \geq 0$. Since $g(\varphi_k)$ is convex with respect to φ_k, by letting its first-order derivative $g'(\varphi_k) = 1 - b\tilde{\Lambda} e^{-\tilde{\Lambda}\varphi_k} = 0$, we have

$$\begin{aligned} \min_{\varphi_k} \varphi_{k+1} = g_{\min}(\varphi_k) &= g\Big(\frac{1}{\tilde{\Lambda}}(\log b + \log \tilde{\Lambda})\Big) \\ &= \frac{1}{\tilde{\Lambda}}(\log b + \log \tilde{\Lambda}) - a + \frac{1}{\tilde{\Lambda}} \\ &= \frac{1}{\tilde{\Lambda}}(\log b + \log \tilde{\Lambda}) + \frac{\alpha}{\alpha - 1}\underline{\tau} \geq 0. \end{aligned}$$

We now verify that for some $R > 0$, if $\varphi_k \in [0, R]$, then $g(\varphi_k) \leq R$. Since g is a convex function, we have $g(\varphi_k) \leq \max\left\{R - a + be^{-\tilde{\Lambda}R}, b - a\right\}$. Thus, we require that $R \geq \max\left\{R - a + be^{-\tilde{\Lambda}R}, b - a\right\}$, *i.e.,* that $R \geq b - a$ and that

$$a \geq be^{-\tilde{\Lambda}R} \quad \Longleftrightarrow \quad \log\left(\frac{a}{b}\right) \geq -\tilde{\Lambda}R,$$

so that $R \geq \frac{-\log(a/b)}{\tilde{\Lambda}}$. For $R \geq \max\left\{b - a, \frac{-\log(a/b)}{\tilde{\Lambda}}\right\}$, g maps $[0, R]$ to itself.

We now show that $|g'(\varphi_k)| < 1$. Since $b > 0$ and $\tilde{\Lambda} > 0$, we see that $g'(\varphi_k) = 1 - b\tilde{\Lambda}e^{-\tilde{\Lambda}\varphi_k} < 1$ for $\varphi_k \geq 0$. To show that $g'(\varphi_k) > -1$, we note that $\min g'(\varphi_k) = g'(\min \varphi_k)$ due to the monotonic increase of $g'(\varphi_k)$. We thus deduce that

$$1 - b\tilde{\Lambda}e^{-\tilde{\Lambda}(\min \varphi_k)} > -1 \quad \Longleftrightarrow \quad \frac{-\alpha}{\alpha - 1}\frac{N}{M}\Lambda\underline{\tau} < \log 2,$$

which always holds since $\alpha > 1$. Thus, the recursion (1) converges to an equilibrium. Furthermore, by setting $\varphi_k = \varphi_{k+1}$ in (1), the result in (3) can be obtained. □

Note that a finite task waiting time φ necessarily implies a finite waiting time for the job as a whole. As a remark, we observe that the condition in (2) is equivalent to $\tilde{\Lambda}^{-1} \geq \alpha\underline{\tau}/(\alpha - 1)$, *i.e.,* the expected time between task arrivals should be larger than the expected runtime, and this necessarily means that the denominator in (3) is positive.

Intuitively, to ensure a finite waiting time, the next task should on average arrive after the previous task has finished running. Proposition 1 implies that finite waiting times are more likely to occur when jobs have a shorter runtime (α is larger) and arrive less often (Λ is smaller). These conditions imply that jobs require fewer resources, allowing the CVSP to rent fewer VMs while still maintaining an upper bound on users' expected waiting times. We can mathematically verify this intuition in the following corollary:

COROLLARY 1. *The waiting time φ, as defined in* (3)*, increases with the instance arrival rate $\tilde{\Lambda}$.*

PROOF. Although Corollary 1 is intuitively true, its proof is not straightforward. To show the monotonic increase of $\varphi(\tilde{\Lambda})$, we prove that, for any $z \geq 1$, $\varphi\left(\frac{1}{z}\tilde{\Lambda}\right)$ decreases with z. Similar to the proof of Proposition 1, we further rewrite $\varphi(z)$ by replacing $a(z) = \frac{z}{\tilde{\Lambda}} - \frac{\alpha\underline{\tau}}{\alpha - 1}$ and $b(z) = \frac{\alpha z}{\tilde{\Lambda}}\left(\frac{1}{z}\tilde{\Lambda}\underline{\tau}\right)^{\alpha}\Gamma(-\alpha, \frac{1}{z}\tilde{\Lambda}\underline{\tau})$ and obtain $\varphi(z) = \frac{1}{\tilde{\Lambda}} \log\left(b(z)a(z)^{-1}\right)^z$. Due to the monotonic increase of the logarithm function, $\varphi(z)$ monotonically decreases if $g(z) = \left(b(z)a(z)^{-1}\right)^z$ monotonically decreases.

By taking the first-order derivative of $g(z)$ with respect to z, we have

$$\frac{\partial g(z)}{\partial z} = \left(\log\frac{b(z)}{a(z)}\right)\left(\frac{b(z)}{a(z)}\right)^z a(z)^2\left(\frac{\partial b(z)}{\partial z}a(z) - b(z)\frac{\partial a(z)}{\partial z}\right),$$

where $\frac{\partial b(z)}{\partial z}a(z) \le b(z)\frac{\partial a(z)}{\partial z}$ is equivalent to

$$\underline{\tau}\left(z - \frac{\alpha}{\alpha-1}\tilde{\Lambda}\underline{\tau}\right)\underline{\tau}^{-\alpha-1}e^{-\frac{1}{z}\tilde{\Lambda}\underline{\tau}} \le \alpha(z-\tilde{\Lambda}\underline{\tau})\int_{\underline{\tau}}^{\infty}\tau^{-\alpha-1}e^{-\frac{1}{z}\tilde{\Lambda}\tau}\mathrm{d}\tau.$$

By leveraging the condition in (2), $\frac{\alpha}{\alpha-1}\frac{1}{z}\tilde{\Lambda}\underline{\tau} \le \log 2$ leads to $\underline{\tau}\left(z - \frac{\alpha}{\alpha-1}\tilde{\Lambda}\underline{\tau}\right) \le \alpha(z-\tilde{\Lambda}\underline{\tau})$. Combing with the fact that $\underline{\tau}^{-\alpha-1}e^{-\frac{1}{z}\tilde{\Lambda}\underline{\tau}} \le \int_{\underline{\tau}}^{\infty}\tau^{-\alpha-1}e^{-\frac{1}{z}\tilde{\Lambda}\tau}\mathrm{d}\tau$, we find that $\frac{\partial b(z)}{\partial z}a(z) \le b(z)\frac{\partial a(z)}{\partial z}$ always holds. □

To illustrate Corollary 1, Figure 3 shows the joint effects of the number of instances M that the CVSP rents and the expected number of instances N that each user will request on the waiting time φ. As expected, the waiting time decreases with the number of rented instances (as shown by the solid curves with grey markers). Also, the instance arrival rate $\tilde{\Lambda} \equiv N\Lambda/M$ decreases as the number of instances M increases. However, the waiting time increases with N, as does $\tilde{\Lambda}$ (as shown by the dashed curves with blue markers). As Λ approaches the upper bound in (2), we see that the waiting time increases rapidly, eventually approaching infinity as predicted by Proposition 1.

3.2.2 System Idle Time and CVSP Viability

The second requirement for the CVSP's viability is to ensure that its expected idle time falls below the threshold required to qualify for a sustained-use discount. Intuitively, less idle time occurs when users submit jobs more frequently and fewer instances have been rented by the CVSP, *i.e.,* the opposite conditions as required for a finite waiting time in Proposition 1. We now find conditions under which both objectives can hold.

We define the *idle-to-runtime ratio* of the system to be the expected ratio of each instance's idle time to the time for which jobs run on the instance. To derive this ratio, we take an integral over the range $\tau \le t$, and arrive at the following proposition:

PROPOSITION 2. *The expected idle-to-runtime ratio is*

$$\theta = \alpha\left(\tilde{\Lambda}\underline{\tau}\right)^{\alpha}\Gamma\Big(-\alpha-1, \tilde{\Lambda}\underline{\tau}\Big). \tag{4}$$

PROOF. When $\tau \le t$, the idle time is $t-\tau$ after a job finishes running and before the next job arrives. So the idle-to-runtime ratio for the system is given by $\frac{t-\tau}{\tau}$. Since τ and t follow the Pareto and exponential distributions respectively, the expected idle-to-runtime ratio is integrated over $\tau \le t$ and $\tau \ge \underline{\tau}$. The proof is readily obtained by evaluating

$$\begin{aligned}&\int_{\underline{\tau}}^{\infty}\left(\int_{\underline{\tau}}^{t}\frac{1}{\tau}(t-\tau)\frac{\alpha\underline{\tau}^{\alpha}}{\tau^{\alpha+1}}\mathrm{d}\tau\right)\tilde{\Lambda}e^{-\tilde{\Lambda}t}\mathrm{d}t\\ &= \int_{\underline{\tau}}^{\infty}\left(\frac{1}{\alpha+1}\underline{\tau}^{\alpha}t^{-\alpha} + \frac{\alpha}{\alpha+1}\underline{\tau}^{-1}t - 1\right)\tilde{\Lambda}e^{-\tilde{\Lambda}t}\mathrm{d}t\\ &= \frac{\alpha}{\alpha+1}\left(\tilde{\Lambda}\underline{\tau}\right)^{-1}e^{-\tilde{\Lambda}\underline{\tau}} - \frac{\alpha}{\alpha+1}\left(\tilde{\Lambda}\underline{\tau}\right)^{\alpha}\Gamma\left(-\alpha, \tilde{\Lambda}\underline{\tau}\right).\end{aligned}$$

Using integration by parts, *i.e.,* $\Gamma(-\alpha, \tilde{\Lambda}\underline{\tau}) = (-\alpha-1)\Gamma(-\alpha-1, \tilde{\Lambda}\underline{\tau}) + (\tilde{\Lambda}\underline{\tau})^{-\alpha-1}e^{-\tilde{\Lambda}\underline{\tau}}$, the above integral can be further reduced to $\alpha(\tilde{\Lambda}\underline{\tau})^{\alpha}\Gamma(-\alpha-1, \tilde{\Lambda}\underline{\tau})$. □

The CVSP qualifies for a sustained-use discount if the idle-to-runtime ratio is sufficiently low. Suppose that the CSP offers sustained-use discounts once the overall usage of each CVSP's instance exceeds a fraction $\epsilon \in (0,1]$ of the total time in an instance billing cycle. Equivalently, the CVSP must satisfy $\frac{1}{\theta+1} \ge \epsilon$, where $\frac{1}{\theta+1}$ is the fraction of time used to run tasks when the instance is either idle or running a task. This condition holds for higher instance arrival rates (larger $\tilde{\Lambda}$):

COROLLARY 2. *The idle-to-runtime ratio θ given in* (4) *decreases as the instance arrival rate $\tilde{\Lambda}$ increases. Furthermore, the CVSP qualifies for a sustained-use discount if the number of instances that it rents satisfies*

$$M \le \left\lfloor\left(\frac{1}{\alpha}+1\right)\left(\frac{1}{\epsilon}-1\right)N\Lambda\underline{\tau}\right\rfloor. \tag{5}$$

PROOF. By taking the first-order derivative of θ with respect to $\tilde{\Lambda}$, we have

$$\begin{aligned}\frac{\partial\theta}{\partial\tilde{\Lambda}} &= \alpha\underline{\tau}(\tilde{\Lambda}\underline{\tau})^{\alpha-1}\big(\alpha\Gamma(-\alpha-1, \tilde{\Lambda}\underline{\tau}) - (\tilde{\Lambda}\underline{\tau})^{-\alpha-1}e^{-\tilde{\Lambda}\underline{\tau}}\big)\\ &\overset{(a)}{=} -\frac{\alpha\underline{\tau}}{\alpha+1}(\tilde{\Lambda}\underline{\tau})^{\alpha-1}\left(\frac{e^{-\tilde{\Lambda}\underline{\tau}}}{(\tilde{\Lambda}\underline{\tau})^{\alpha+1}} + \alpha\Gamma(-\alpha, \tilde{\Lambda}\underline{\tau})\right)\\ &\le 0,\end{aligned}$$

where (a) is due to $\Gamma(-\alpha, \tilde{\Lambda}\underline{\tau}) = (-\alpha-1)\Gamma(-\alpha-1, \tilde{\Lambda}\underline{\tau}) + (\tilde{\Lambda}\underline{\tau})^{-\alpha-1}e^{-\tilde{\Lambda}\underline{\tau}}$. Since θ decreases with $\tilde{\Lambda}$, we then need to prove $\theta \le \frac{1}{\epsilon} - 1$ at $\tilde{\Lambda} = 0$. By substituting $\lim_{\tilde{\Lambda}\to 0}\Gamma(-\alpha-1, \tilde{\Lambda}\underline{\tau}) = \frac{1}{\alpha+1}(\tilde{\Lambda}\underline{\tau})^{-\alpha-1}$ and combining with $\theta \le \frac{1}{\epsilon} - 1$, we have $\theta \le \frac{\alpha}{\alpha+1}\left(\frac{N}{M}\Lambda\underline{\tau}\right)^{-1} \le \frac{1}{\epsilon} - 1$, which leads to (5). □

We thus find that the CVSP is viable if the conditions in Proposition 1 and Corollary 2 are simultaneously satisfied:

$$\left\lceil\frac{\alpha N}{\alpha-1}\Lambda\underline{\tau}\right\rceil \le M \le \left\lfloor\left(\frac{1}{\alpha}+1\right)\left(\frac{1}{\epsilon}-1\right)N\Lambda\underline{\tau}\right\rfloor, \tag{6}$$

so that users experience finite waiting times and the CVSP qualifies for a sustained-use discount. Indeed, we find that (6) has a solution when $\frac{\alpha}{\alpha-1} \le \frac{(\alpha+1)(1-\epsilon)}{\alpha\epsilon}$, leading to

COROLLARY 3. *There exists an instance arrival rate $\tilde{\Lambda}$ for which (6) is satisfied,* i.e., *the CVSP is viable, if*

$$\alpha \ge \sqrt{\frac{1-\epsilon}{1-2\epsilon}}, \tag{7}$$

which has a positive solution α only if $\epsilon < \frac{1}{2}$.

Since the expected runtime decreases as α increases, we conclude that jobs with shorter runtimes are more suitable for the CVSP. Intuitively, if the majority of users' job runtimes are sufficiently small, then even if many jobs are submitted to the system, each job will not have to wait for very long. Thus, the CVSP can maintain low waiting times and low idle-to-runtime ratios, thus qualifying for a sustained-use discount, by attracting smaller jobs. Indeed, these smaller jobs are exactly those that ought to be attracted to the CVSP; jobs with longer runtimes naturally qualify for a

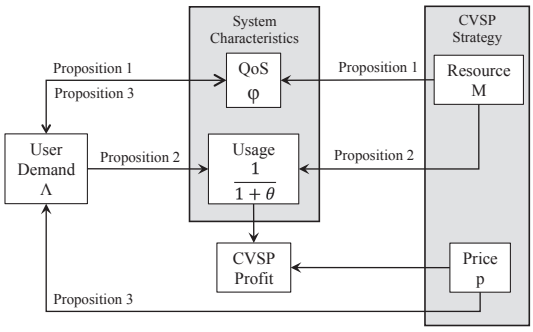


Figure 4: Tradeoffs between different parameters, showing their economic and behavioral impacts.

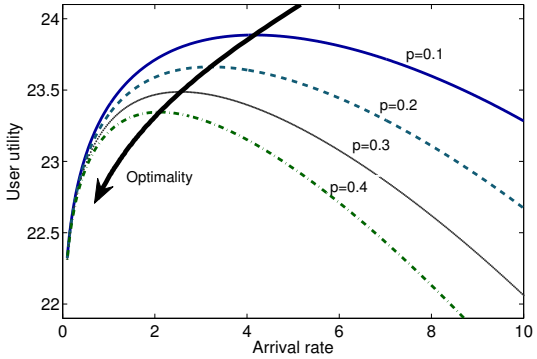


Figure 5: Example of the utility-maximizing demand Λ decreasing with price p for a single user.

sustained-use discount at the CSP and need not make use of the CVSP.

However, if the sustained-use threshold ϵ is too large, shorter average runtimes are not enough to ensure finite job waiting times. As ϵ grows, the CVSP must attract more and more jobs to lower the system idle time; after a certain point, jobs will need to wait for too many preceding jobs to finish, and the waiting time will grow to infinity despite the short job runtimes. While we do not expect the threshold of $\epsilon = 0.5$ in Corollary 3 to hold exactly, this result indicates that even relatively generous thresholds, like $\epsilon = 75\%$ utilization, may not allow for CVSP viability.

4. THE OPTIMAL CVSP STRATEGY

By leveraging the characteristics of the CVSP workload derived in Section 3, we can now derive the users' and the CVSP's optimal strategies. Both must trade off between possibly conflicting objectives: users would like both low waiting times and low instance prices, while the CVSP desires as little idle time as possible in order to qualify for a larger sustained-use discount. The user and CVSP decisions can be broken down as follows:

- *User demand (job arrival rates)*: Users' demands are dictated by their job arrival rates λ_l, $l = 1, ..., L$, which determine the frequency of job submission. Intuitively, users' demands increase as their waiting times φ decrease and the CVSP's price p drops. We assume that users adjust their demands to maximize their expected utilities, which are a function of these factors.

- *Waiting time (user QoS)*: Though each user's demand λ_l is expected to decrease as the waiting time φ increases, φ increases with the system arrival rate Λ (Corollary 1), which paradoxically corresponds to a higher demand λ_l from user l. Moreover, each user's demand depends on the other users' demands via the waiting time φ. Given this dependence, we find an equilibrium system arrival rate at which all of the users can simultaneously maximize their utilities. Since φ also depends on M, the number of instances rented by the CVSP, this equilibrium demand indirectly depends on M.

- *CVSP profit (idle-to-runtime ratio)*: We suppose that the CVSP's objective is to maximize its profit, which depends on the idle-to-runtime ratio θ: a lower θ (less idle time) leads to the CVSP obtaining a higher sustained-use discount from the CSP. The CVSP can influence θ through two decision variables: the price p that it charges users, which in turn influences users' demands Λ, and the number of instances M that it rents from the CSPs. Intuitively, a smaller M can lead to a lower θ, but may also lead to a longer waiting time and thus lower user demand. Therefore, different values of M and p can lead to lower or higher profit.

Figure 4 visualizes the above discussion (Proposition 3 is given in Section 4.1).

In the rest of this section, we first derive a relationship between the user demand, the CVSP price, and the expected waiting time (Section 4.1). We then formulate the CVSP's profit maximization problem (Section 4.2), which the CVSP solves to determine the optimal M and p.

4.1 User Demands

We first derive users' optimal demands by considering their utilities. We define an overall utilization rate for each user as the ratio of his or her expected job runtime to the expected waiting time, multiplied by the user's job arrival rate and the number of tasks per job, *i.e.,* $r_l = \lambda_l \frac{\alpha \underline{\tau} N}{(\alpha-1)\varphi}$. User utility generally increases with this rate, but with diminishing returns (*i.e.,* only increasing modestly when the runtime is already much larger than the waiting time). As such, we model this using the standard concave utility function $\log(1 + r_l)$ [20, 39], with the lth user's utility given by:

$$U(\lambda_l \,|\, p) = \gamma_l \log\left(1 + \lambda_l \frac{\alpha \underline{\tau} N}{(\alpha-1)\varphi}\right) - p\lambda_l \frac{\alpha \underline{\tau} N}{\alpha - 1}. \qquad (8)$$

Here, $p\lambda_l \frac{\alpha \underline{\tau} N}{\alpha-1}$ is the expected amount that user l pays to the CVSP given its expected job runtime $\frac{\alpha \underline{\tau}}{(\alpha-1)}$, and $\gamma_l > 0$ is a normalization constant that encodes the user's relative utilities from its waiting time and payment to the CVSP.

Note that the waiting time φ depends on *all* users' job arrival rates. Thus, due to λ_l's contribution to the waiting time, the utility in (8) for user l may not be concave in user l's demand λ_l. However, we see in Figure 5 that user l's utility does first increase and then decrease in λ_l. For instance, given a price $p = \$0.1$, the user's maximum utility occurs at $\lambda_l = 4.2$, corresponding to a waiting time of 0.046 hours. Higher arrival rates decrease user utility due to (i) higher

costs and (ii) longer expected waiting times. User utility can therefore be maximized by setting $\partial U(\lambda_l \,|\, p)/\partial \lambda_l = 0$.

We now derive the system's job arrival rate $\Lambda = \sum_{l=1}^{L} \lambda_l$ assuming that all users maximize their own utilities, *i.e.,* $\partial U(\lambda_l \,|\, p)/\partial \lambda_l = 0$ for all $l = 1, \ldots, L$. We find that each user's utility-maximizing arrival rate satisfies $\gamma_l \varphi - p\varphi^2 = \lambda_l(\gamma_l \frac{\partial \varphi}{\partial \Lambda} + p\frac{\alpha\tau}{\alpha-1} N\varphi)$, leading to the following equilibrium system arrival rate:

PROPOSITION 3. *If all users choose the job arrival rates that maximize their utilities, then the system arrival rate satisfies*

$$\Lambda = \sum_{l=1}^{L} \frac{\gamma_l \varphi - p\varphi^2}{\gamma_l \frac{\partial \varphi}{\partial \Lambda} + p\frac{\alpha\tau}{\alpha-1} N\varphi}. \tag{9}$$

We thus find that Λ decreases for higher prices: if the CVSP charges users a higher price p, they are less willing to endure larger job waiting times, leading to a lower demand. We illustrate in Figure 5 that the utility-maximizing Λ decreases with p, as shown in the bolded black arrow curve.

4.2 Profit Maximization

We now characterize the CVSP's sustained-use discount in terms of the idle-to-runtime ratio θ. Letting π denote the CSP's full unit price without discounts, we expect the discount to increase as θ decreases, *i.e.,* the unit price should be cheaper when the CVSP has less idle time. Letting $1-\omega$ denote the discount, *i.e.,* ω is the fraction of the full price that the user must pay, we define ω as a linear function of the idle-to-runtime ratio θ:

$$\omega(\theta) = \delta_1 \theta + \delta_2, \tag{10}$$

where $\delta_1 > 0$ and $\delta_2 > 0$ are two small fractions, satisfying $\omega(\theta) \in (0, 1]$ for any $\theta \leq \frac{1-\epsilon}{\epsilon}$. This threshold corresponds to $\epsilon \leq \frac{1}{1+\theta}$, *i.e.,* the fraction of time during which some tasks are running should be larger than ϵ. In Appendix A, we give a real-world example of (10) for Google Cloud Platform.

The CVSP's objective in choosing how many instances to rent and how much to charge is to maximize its own profit. The monthly payment to CSPs and the revenue received from users are $\omega(\theta)\pi\frac{T}{\theta+1}$ and $p\frac{T}{\theta+1}$ respectively, where T represents a month's time, and $\frac{T}{1+\theta}$ corresponds to the expected amount of time that a job is running on each instance. The CVSP thus solves the following optimization problem to maximize its profit:

$$\text{maximize} \quad M\Big(p - \omega(\theta)\pi\Big)\frac{T}{\theta+1} \tag{11a}$$

$$\text{subject to} \quad \omega(\theta)\pi \leq p \leq \pi \tag{11b}$$

$$\omega(\theta) = \delta_1\theta + \delta_2 \tag{11c}$$

$$\left\lceil \frac{\alpha N}{\alpha-1}\Lambda\underline{\tau} \right\rceil \leq M \leq \left\lfloor \frac{\alpha+1}{\alpha}\frac{1-\epsilon}{\epsilon} N\Lambda\underline{\tau} \right\rfloor \tag{11d}$$

$$\text{variable:} \quad M \in \mathbb{Z}^+, p \in \mathbb{R}^+, \tag{11e}$$

where (11b) ensures that the price is lower than the full price and higher than the price with sustained-use discount, so that users are motivated to use the CVSP in the first place and the CVSP can have positive profit. The left-hand side of (11d) ensures a finite waiting time as in Proposition 1, and the right-hand side ensures a usage larger than ϵ as stated in Corollary 2. The waiting time φ, idle-to-runtime ratio θ, and equilibrium system demand Λ used in (11) are respectively given in (3), (4), and (9).

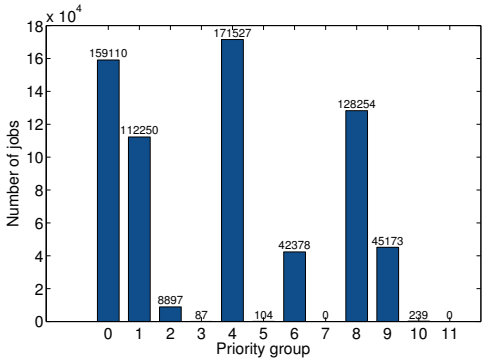


Figure 6: Numbers of jobs in different priority groups for our dataset.

If a joint solution to (3), (4), and (9) can be found subject to the constraints (11b–11d), then the profit maximization (11) has a solution. We expect that such a solution does exist, since these three equations can be reduced to one nonlinear equation in p and M. Although (11) is a discontinuous, mixed integer, nonlinear programming problem, it can be numerically solved by exhaustively considering all possible values of M and doing a line search for the optimal p.[5] In the next section, we analyze how the CVSP profits from different combinations of M and p. This not only provides insights into the interactions between the CVSP's strategy and user behavior, but also demonstrates that there is a feasible set for (11), *i.e.,* that the CVSP can make money in the IaaS market.

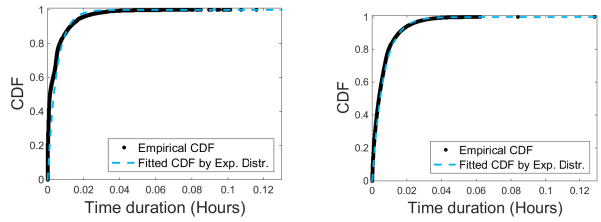
5. DATA-DRIVEN EVALUATION

To verify our models and results, we employ a trace from Google's production compute cluster, spanning 672,003 jobs over roughly a month-long period in May 2011 [34]. We begin in Section 5.1 by analyzing the job arrival and runtime distributions, showing that they match our assumptions within a tolerable margin of error. Then, in Section 5.2, we use simulations to validate the system characterization and profit maximization derived in Sections 3 and 4, respectively. Overall, we illustrate that all three parties – CSPs, CVSP, and users – can benefit from an optimal cloud resource reselling mechanism.

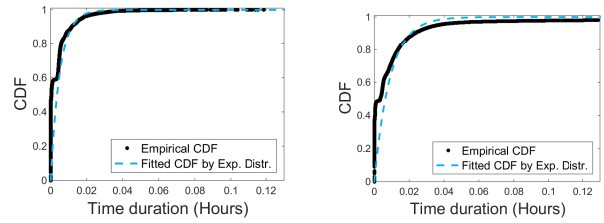
5.1 Model Validation

Data trace. The user trace contains timestamp records of machine events, job events, and task events; as the resource utilization of different jobs was studied in [26,38], we do not consider these parameters in our analysis. When a job is submitted, the user can specify its priority as one of 12 integer groups from 0 to 11. In general, a higher numbered group will be used for a more urgent job, while lower-prioritized jobs may be, *e.g.,* remote monitoring products or backend batch processing. Since jobs' priorities will affect their waiting times and runtimes, *e.g.,* due to higher-priority

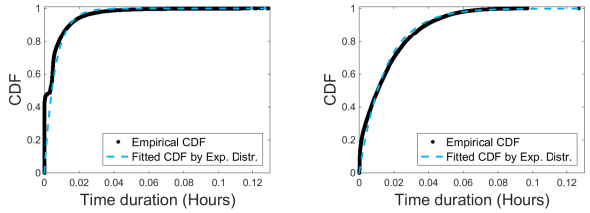
[5]Since the value of M is at most 1000 for all practical purpose, (11) can be solved in a reasonable amount of time.



(a) Priority group 0 ($\Lambda = 200$, p-value $= 0.0015$). (b) Priority group 1 ($\Lambda = 150$, p-value $= 0.1998$).



(c) Priority group 4 ($\Lambda = 195$, p-value $= 0.001$). (d) Priority group 6 ($\Lambda = 110$, p-value $= 0.001$).



(e) Priority group 8 ($\Lambda = 160$, p-value $= 0.001$). (f) Priority group 9 ($\Lambda = 66$, p-value $= 0.0625$).

Figure 7: Comparison between the empirical and fitted exponential CDFs of the duration between job submissions for different priority groups. The curves qualitatively match in each case, and the priority groups 0, 1 and 9 have non-rejected p-values.

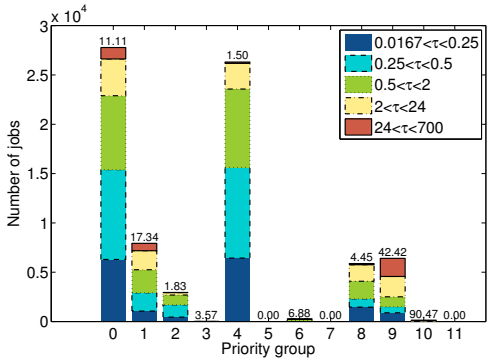
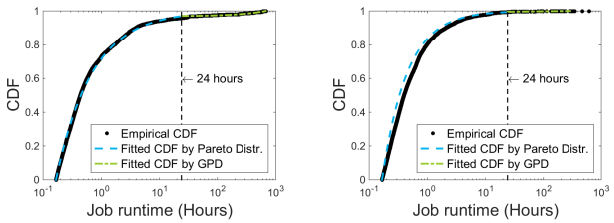


Figure 8: Distribution of the job runtime in each priority group. The number on each bar indicates the average runtime in the priority group.



(a) Priority group 0 ($\alpha = 0.71$, $\xi = 2.14$, $\sigma = 21187.83$, $\mu = 27.98$). (b) Priority group 4 ($\alpha = 1.01$, $\xi = 524.18$, $\sigma = 5037.03$, $\mu = 0.96$).

Figure 9: Comparison of the empirical and piecewise-fitted CDFs of the job runtime distributions. Qualitatively, the CDFs appear to be very similar for both groups. The R-squared fit statistics are $R^2 = 0.997, 0.872$ for the Pareto and Generalized Pareto (GPD) fits respectively for priority group 0; $R^2 = 0.9913, 0.9319$ for the Pareto and GPD fits respectively for priority group 4.

jobs being allocated more resources faster, we consider the priority groups separately.

Figure 6 shows the number of jobs in each priority group.[6] For brevity, we show only the results for the most used groups in the body of the paper. Overall, we find that the job arrival and runtime distributions are comparable to the Poisson and Pareto distributions assumed in Section 3.

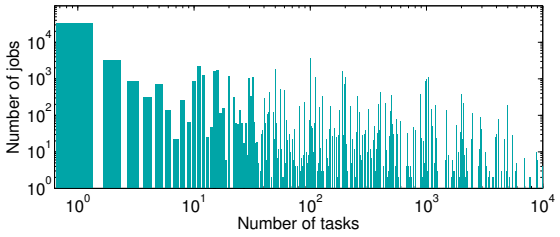
Poisson-distributed arrival rates. We validate our assumption of Poisson arrivals by examining the distributions of job interarrival times for the priority groups with substantial sample sizes, as shown in Figure 7. In all of the plots, the black dotted line gives the empirical cumulative density function (CDF), and the blue dashed line represents the best-fit CDF from an exponential distribution. Visually, the curves are qualitatively similar in each case. To investigate these fits statistically, we use the Lilliefors test on the null hypothesis that the waiting times come from an exponential distribution, at the significance level of 0.1% (*i.e.,* if the test returns a p-value ≤ 0.001, it rejects the hypothesis of an exponential distribution). The resulting p-values are given in the captions; the Lilliefors test does not reject the hypothesis for the jobs in priority groups 0, 1, and 9, though it does for the jobs in priority groups 4, 6, and 8. The CDFs for priority groups 4, 6, and 8 in Figure 7(c), 7(d), and 7(e) have a small kink around a 20-second interarrival time, suggesting that Google may impose a small minimum delay on jobs whose tasks are not scheduled simultaneously. The kink is likely the reason that the p-value for this group does not reject the null hypothesis.

Pareto-distributed job runtimes. Figure 8 depicts the number of jobs with various runtimes in each priority group, identified by the stacked color bars.[7] We notice that the longest running jobs, *i.e.,* those running for more than one day, as shown in the red bars with solid outlines, are mostly in either the highest or lowest priority groups. Jobs with lower priority may be long-running computational jobs, which are not latency-sensitive, while higher priority jobs may be real-time, long-term applications that cannot tolerate interruptions.

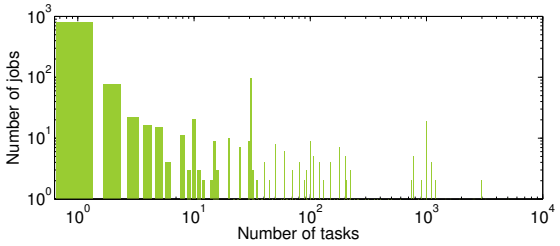
As with the arrival rates, we choose to examine only the

[6]We exclude jobs submitted before the beginning of the trace as we do not know their runtimes or arrival times.

[7]The total number of jobs in Figure 8 is less than that shown Figure 6 because we additionally exclude the jobs that were unable to finish within the trace period; we cannot determine the total runtimes of those jobs.



(a) Usage $\frac{1}{\theta+1} < 0.25$ (average $N = 192$).



(b) Usage $\frac{1}{\theta+1} \geq 0.25$ (average $N = 67$).

Figure 10: Number of jobs comprising different numbers of tasks in different ranges of job runtimes. Those with shorter runtimes (10(a)) have more tasks than those with longer runtimes (10(b)).

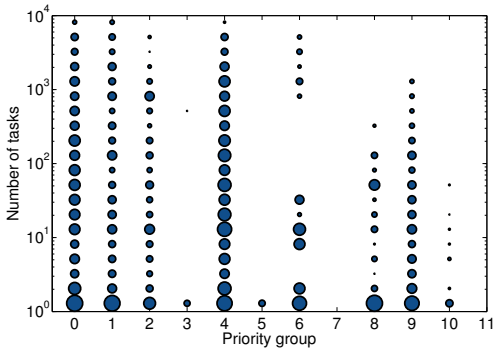


Figure 11: The number of jobs in the priority groups that have comprise numbers of tasks in given ranges. A larger circle implies that more jobs in that priority group have that number of tasks, with the size of dots in logarithmic scale. Most jobs have fewer than 100 tasks.

runtimes of priority groups 0 and 4, which have the most jobs in Figure 8. In each plot of Figure 9, the empirical CDF, denoted by the dotted black lines, evinces a power-law distribution at the beginning with a more complex tail. To incorporate the dual nature of the distribution in fitting the CDF, we use the following piecewise probability density function that makes the first segment a Pareto distribution and the tail a generalized Pareto distribution (similar to the process used for job stragglers in [28]):

$$f(\tau) = \begin{cases} \frac{\alpha \underline{\tau}^{\alpha}}{\tau^{\alpha+1}}, & \text{if } \underline{\tau} \leq \tau \leq \hat{\tau}, \\ \frac{1}{\sigma}\left(1 + \xi\frac{\tau-\mu}{\sigma}\right)^{-\left(\frac{1}{\xi}+1\right)}, & \text{if } \tau \geq \hat{\tau}, \end{cases}$$

where the CDF is $F(\tau) = 1 - (\underline{\tau}/\tau)^{\alpha}$ for $\underline{\tau} \leq \tau \leq \hat{\tau}$ and $F(\tau) = 1 - (\underline{\tau}/\hat{\tau})^{\alpha} + \left(1 + \xi\frac{\hat{\tau}-\mu}{\sigma}\right)^{-\frac{1}{\xi}} - \left(1 + \xi\frac{\tau-\mu}{\sigma}\right)^{-\frac{1}{\xi}}$ for $\tau \geq \hat{\tau}$. Setting the domain boundary at $\hat{\tau} = 24$ hours, we fit the parameter α for the Pareto distribution (the dashed blue curves in Figure 8) and the parameters σ, ξ, and μ for the generalized Pareto distribution (the dash-dot green lines). We observe that the fitted CDF fits the empirical data well visually, and quantitatively, all of the R-squared fit statistics are above $R^2 = 0.87$. Although α in Figure 9(a) is less than 1, implying an infinite expected runtime in theory, it is close to 1; since all runtimes are in practice finite, we assign a value to α that is slightly larger than 1 to simulate the job runtimes in Section 5.2.

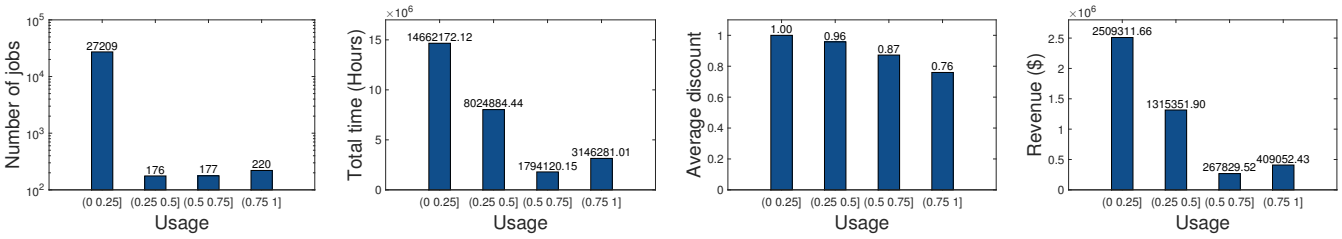
Expected number of tasks. Another important property of the jobs is the number of tasks; as in Section 3, N can affect the instance arrival rate (cf. Lemma 1), and henceforth the expected waiting time (cf. Proposition 1) and the idle-to-runtime ratio (cf. Proposition 2). Moreover, jobs that require more tasks than the CVSP's number of rented instances M will not be able to run on the CVSP. The average number of tasks for all jobs recorded in the trace is 190, though a large majority of jobs comprise fewer than 100 tasks.

In Figure 10, we show the histograms for the number of tasks per job, over jobs of all priorities. Since Google offers sustained-use discounts when the job usage exceeds one quarter of a month (cf. Appendix A), we consider jobs above and below this usage threshold separately. As expected, jobs with shorter runtimes (in Figure 10(a)) have more tasks than the jobs with longer runtimes (in Figure 10(b)): jobs that can be divided into more tasks are more likely to have lower per-task runtimes. Thus, some jobs with short runtimes may wish to use the CVSP in order to receive a sustained-use discount, but will not be eligible to do so if the CVSP does not have enough instances. We show in the next section that it is optimal for the CVSP to rent only a limited number of instances, forcing these jobs to use services from the CSP instead and allowing both the CVSP and CSP to coexist in the IaaS market.

Figure 11 shows the numbers of tasks for jobs in each of the priority groups. We observe that, as we would expect from Figure 10, most jobs have fewer than 100 tasks. There is little visual consistency between the numbers of tasks for jobs with different priorities, but priority groups 0 and 4 clearly have the most jobs of all the priority groups, as is consistent with Figure 8. Interestingly, both of these priority groups appear to have somewhat more jobs that require a large number of tasks. We conjecture that priority groups with more jobs are more likely to include relatively rare jobs that have large numbers of tasks.

5.2 Trace-based Simulations

We use jobs from the trace that belong to priority group 0 to simulate the behavior of the CSPs, CVSP, and users. We first consider the CSP's revenue from these jobs with sustained-use discounts, before the CVSP enters the market. We then consider the CVSP's revenue and user utility after the CVSP enters the market. This trace was taken before Google introduced sustained-use discounts, so we cannot fully measure the effect of the sustained-use discounts in practice, but the job workload data still allow us to approximate its anticipated effects.



(a) Number of jobs. (b) Total time. (c) Average fraction paid. (d) Revenue.

Figure 12: The number of jobs, the total time for which users need to pay, the average fraction paid after the sustained-use discount, and the revenue for the CSP before the CVSP enters the market. All quantities are broken out for jobs whose usage is in a different range of runtimes.

5.2.1 CSP Revenue without the CVSP

We take the full price offered by the CSP to be $p =$ \$0.17, which is in between the current prices that Google charges for different instance types [14]. Assuming that the users of the CSP acted so as to maximize their utilities, we can then use the workload trace to reverse-engineer the users' utility parameters. From (8), we write $U(\lambda_l \,|\, p) = \gamma \log\left(1 + \lambda_l \frac{\alpha\tau N}{(\alpha-1)\varphi}\right) - p\lambda_l \frac{\alpha\tau}{\alpha-1} N$, assuming that all users have the same γ parameter for simplicity. As in Proposition 3, we can solve for the system arrival rate as

$$\Lambda = \frac{L(\gamma - p\varphi)}{p\frac{\alpha\tau}{\alpha-1}N}. \tag{12}$$

We take the system arrival rate to be $\Lambda = 200$, as discovered in Figure 7(a). We also find from the workload trace that the expected number of tasks per job is $N = 138$, with $L = 306$ unique users submitting jobs. Furthermore, the jobs experience an average delay of $\varphi = 6.85 \times 10^{-3}$ hours between the time when they are submitted and the time they begin to run. We can then use (12) to find that $\gamma = 9$.

Given these user utilities, we now calculate the CSP's revenue under the sustained-use discounts, assuming no change in user demand (*i.e.,* the CVSP has not yet entered the market). We suppose that the CSP follows the pricing discounts specified in Appendix A. In Figure 12(a), we count the number of jobs that fall into each runtime range of the piecewise discount function, *e.g.,* jobs falling into the $(0, 0.25]$ range run for less than one-quarter of the month discovered in the dataset. We can then obtain the total instance hours consumed by the jobs in each range by calculating $\sum_{i=\in\{\text{All jobs}\}} N_i\tau_i$, as shown in Figure 12(b). Though there are about the same numbers of jobs in the ranges $(0.25, 0.5]$, $(0.5, 0.75]$, and $(0.75, 1]$, those in the range $(0.25, 0.5]$ have a much larger number of instance hours, suggesting that these jobs have a larger average number of tasks. In Figure 12(c), we show the average sustained-use discount for all jobs in different usage ranges, and we finally obtain the revenue of the CSP from jobs in each range in Figure 12(d). Jobs in the range $(0, 0.25]$ each run for less than a quarter of the month, so we expect them to have relatively low per-job revenue. However, there are enough jobs in this range for them to dominate the CSP's total revenue.

This result raises the question of whether the CSP would allow the CVSP to exist in the market: given that the CSP makes most of its revenue from shorter jobs, it might be reluctant to allow these jobs migrate to the CVSP. However, the CVSP will not be able to handle all of these jobs: it can only admit jobs whose number of tasks N does not exceed M, the number of instances rented by the CVSP. Thus, by limiting the CVSP's number of instances, the CSP can limit its revenue loss. Indeed, the CVSP itself has an incentive to keep its number of instances limited, since a larger value of M will increase the idle time on each instance, decreasing the sustained-use discount that the CVSP receives (cf. the bounds in (6)). We show in the next section that it is optimal for the CVSP to rent only a few hundred instances. Losing this limited number of jobs can in fact increase the CSP's profit, reducing its per-job account tracking and billing costs, as verified later in Figure 14. The CVSP's lower unit price might even attract more user job submissions than the CSP could attract on its own, increasing the CSP revenue overall due to higher user demand.

5.2.2 CVSP Profit and Impact on CSP Revenue

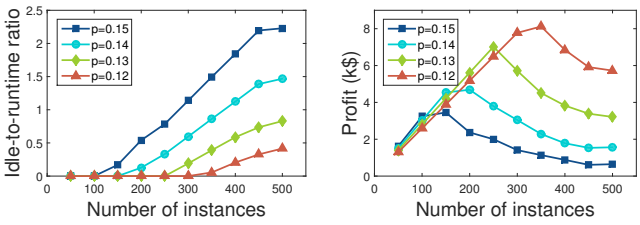
We now show that the CVSP makes a positive profit. We assume that users will offload jobs to the CVSP as long as (i) the price offered by the CVSP is cheaper than what the users can obtain from their CSPs and (ii) the CVSP has enough instances to accommodate the job's tasks. Thus, the number of jobs that the CSPs will lose depends on the CVSP's price p and the number of instances M. While in reality user demand may increase due to the CVSP's lower prices, we consider a conservative scenario in which jobs merely defect from the CSP, so the overall demand does not increase. Indeed, we show in the next section that user utilities do not change significantly when the price changes: users' utilities are much more sensitive to the waiting time and number of instances M than they are to the price.

Table 3 summarizes the number of users L who have at least one job that defects to the CVSP as well as their average number of tasks per job N and the resulting system arrival rate of the CVSP Λ. We see that Λ increases as the number of rented instances M increases and as the price p decreases: more instances allows the CVSP to support jobs with more tasks, and a lower price means more jobs can lower their costs by defecting to the CVSP. As we would expect, the average number of tasks per job N also increases with the number of rented instances, but it decreases as the price p decreases. As p decreases, jobs with longer runtimes can lower their costs by defecting to the CVSP, as the lower price p at the CVSP becomes even lower than their sustained-use discounts from the CSP. These jobs are likely to have fewer tasks (cf. Figure 10).

In Figure 13, we plot the idle-to-runtime ratio and CVSP profit as the CVSP's decision points M and p are varied.

Table 3: Variation of user behavior with price incentive p and rented instances M.

M	$p = 0.160$			$p = 0.145$			$p = 0.130$		
	L	N	Λ	L	N	Λ	L	N	Λ
100	257	1.0550	40.77	260	1.0547	41.36	261	1.0545	41.65
200	260	1.0737	41.28	264	1.0733	41.87	265	1.0731	42.16
300	260	1.0983	41.79	264	1.0981	42.39	265	1.0979	42.69
400	261	1.1048	41.83	265	1.1045	42.44	266	1.1043	42.74
500	266	1.1382	42.07	270	1.1377	42.68	271	1.1376	42.98



(a) Idle-to-runtime ratio. (b) CVSP Profit.

Figure 13: Idle-to-runtime ratio and profit as the CSVP varies its price p and the number of rented instances M. As p and M increase, the idle-to-runtime ratio increases monotonically. The profit is non-monotonic, necessitating a numerical search for the optimal p and M that solve (11).

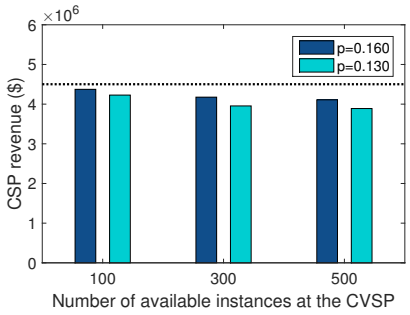
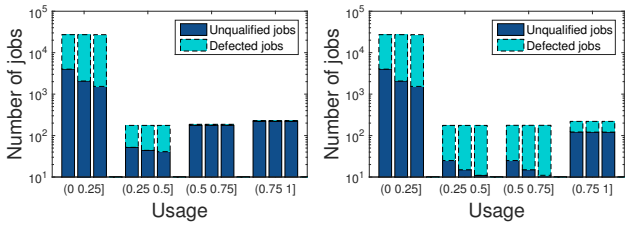


Figure 14: CSP revenue after jobs defect to the CVSP. The dashed line represents its revenue before the CVSP enters the market.



(a) $p = 0.160$. (b) $p = 0.130$.

Figure 15: Jobs defecting to the CVSP for different usage ranges, price p, and number of rented instances M. The left, center, and right bars in each group corresponds to $M = 100, 300, 500$ respectively. A larger M and lower p lead to more jobs defecting.

The minimum price, $p = \$0.12$, represents the maximum sustained-use discount offered by the CSP; the CVSP will not offer a price lower than \$0.12 as in that case its revenue would be negative. We observe in Figure 13(a) that the system has less idle time with a smaller number of instances and a lower price: as the price p decreases, Λ increases, yielding fewer available resources and less idle time. In all cases, the idle-to-runtime ratio θ is ≤ 3, indicating that $\frac{1}{1+\theta} \geq 0.25$ and the CVSP can qualify for a sustained-use discount, earning a positive profit.

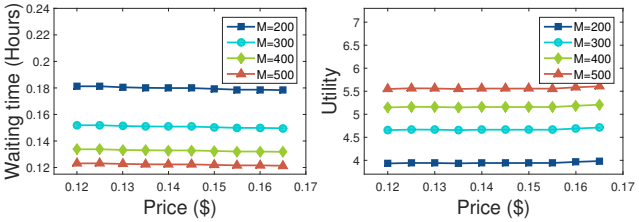
We confirm this reasoning in Figure 13(b). The CVSP's profit, however, is not monotonic: Figure 13(b) shows that if relatively few instances are available ($M < 300$), then a higher price p can yield higher CVSP profit as in (11). For a small number of instances, the CVSP cannot support as many jobs without increasing users' expected waiting time, making a lower demand (as driven by higher price) desirable. However, even though less idle time can increase the CVSP's sustained-use discount from the CSP, the lower arrival rate caused by the fewer available resources can drive the profit down. Numerically, we find that the CVSP's maximum profit is roughly \$8,000, for $M = 350$ and $p = \$0.12$. Thus, the CVSP limits itself to only a few hundred instances; we show in the next section that this represents only a limited number of user jobs, indicating that the CSP loses relatively little revenue. The CSP thus has little incentive to drive the CVSP out of the market.

Figure 14 compares the CSP revenue before and after the CVSP enters the market, for selected values of p and M. Though the CSP loses some revenue, it retains more than 85% of its original revenue, as represented by the dashed black line. The CSP could easily compensate for this small revenue loss. For instance, offloading shorter jobs to the CVSP reduces the CSP's cost of tracking those user accounts and billing users for those jobs. The CSP may also benefit from an increase in user demand at the CVSP, due to users taking advantage of lower CVSP prices and submitting more jobs than they would otherwise have submitted to the CSP.

5.2.3 *User Utility at the CVSP*

We now analyze users' utilities at the CVSP after the CVSP enters the market. As in Table 3, we again assume that users will offload jobs to the CVSP as long as the price offered is cheaper than what the users can obtain from their CSPs and the CVSP has enough instances to accommodate the job's tasks.

Figure 15 shows the number of jobs that defect from the CSP to the CVSP for different prices p and numbers of instances M. We observe that only a limited number of jobs defect to the CSP. When the price is relatively high, at $p = \$0.16$, longer-running jobs do not defect to the CVSP,



(a) Waiting time. (b) Utility.

Figure 16: User waiting time φ and utility U for different prices p and rented instances M. Both are relatively flat with respect to price, but φ decreases and U increases with the number of instances M.

as they can qualify for the sustained use discounts on their own. Even some jobs with shorter running times, *i.e.,* usage in the range $(0, 0.25]$, do not defect, as they have more tasks than the CVSP has instances. More jobs defect when the price is cheaper and when the CVSP rents more instances, since under these conditions more jobs can save money at the CVSP and are eligible (*i.e.,* have few enough tasks) to run at the CVSP, respectively.

Figure 16 shows the average waiting time and utility for users at the CVSP, given the defected jobs as calculated in Figure 15. We observe that neither the waiting time nor utility changes very much with price. This result suggests that user utility, and thus user demand, is largely determined by the waiting time. This finding is consistent with the relatively large weight $\gamma = 9$ that we derived from (12) for the waiting time component of user utility. Figure 16(a) also illustrates that more rented instances M intuitively yield a shorter waiting time (Corollary 1). Consequently, user utility increases with the number of rented instances, as shown in Figure 16(b).

6. CONCLUSION

CVSPs can take advantage of sustained-use IaaS discounts to reduce cloud computing costs for users with short-term jobs, without a significant decrease in CSP revenue. In this work, we give conditions under which the CVSP's business model is viable: we show that sustained-use discount thresholds that are too large, as well as job runtimes that are too long, result in job waiting times growing to infinity at the CVSP. We use our characterization of the CVSP's job scheduling to find users' optimal demands at the CVSP in order to maximize a utility function of the job waiting times, the CVSP price, and the user demand. This determines the CVSP's optimal profit-maximizing price, as well as the optimal number of instances that it should rent from the CSP. We verify our job workload assumptions with a month-long Google datacenter trace and show that it is optimal for the CVSP to rent only a limited number of instances from the CSP. Thus, we obtain a win-win situation: the CVSP can earn a positive profit, and users can save money, without significantly impacting the CSP revenue.

As a first investigation into the viability of CVSPs, our work does not incorporate all practical features of the IaaS market. In particular, we do not explicitly model the CSP's optimal behavior, *i.e.,* the sustained-use discounts it should offer so as to maximize its own profit. Since multiple CSPs may offer different sustained-use discounts in order to attract jobs from their competitors, this results in the variety in formulating CVSP strategy depending on the deal with each CSP. We expect that including this CSP competition can lead to a more in-depth understanding of the CVSP's role in the IaaS market. We also plan to investigate the robustness of our findings to our assumptions of Poisson job arrivals and Pareto runtime distributions.

Acknowledgments

The work in this paper was in part supported by the NSF Waterman Award Grant CNS-1347234, and the Research Grants Council of Hong Kong under Projects No. RGC 11207615.

APPENDIX

A. SUSTAINED-USE DISCOUNTS

Users can get a better deal from their CSPs for long-term commitments. Google Cloud Platform offers no discount for the first quarter of usage, and then 20%, 40%, and 60% off for the second, third and fourth quarter of the monthly usage respectively [14]. For instance, at the third quarter of the usage, the discount is calculated as $\big(0.25 \times 1 + 0.25 \times 0.8 + (\frac{1}{\theta+1} - 0.5) \times 0.6\big)/\frac{1}{\theta+1}$ for $0.5 < \frac{1}{\theta+1} \leq 0.75$. Thus, we have the following sustained-use discount expression to model the Google Cloud Platform:

$$\omega(\theta) = \begin{cases} 1, & \theta \geq 3 \\ 0.05\theta + 0.85, & 1 \leq \theta < 3 \\ 0.15\theta + 0.75, & \frac{1}{3} \leq \theta < 1 \\ 0.3\theta + 0.7, & 0 \leq \theta < \frac{1}{3}. \end{cases}$$

which is consistent with the formulation in (10). However, in this case, the sustained-use discount is piecewise linear.

2. REFERENCES

[1] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. The rise of RaaS: the resource-as-a-service cloud. *Communications of the ACM*, 57(7):76–84, 2014.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. of NSDI*, 2010.

[3] Amazon EC2. Reserved Instances, 2015. https://aws.amazon.com/ec2/purchasing-options/reserved-instances/.

[4] Amazon EC2. Spot Instances, 2015. http://aws.amazon.com/ec2/spot/.

[5] M. Andrews, S. Antonakopoulos, and L. Zhang. Energy-aware scheduling algorithms for network stability. In *Proc. of IEEE INFOCOM*, 2011.

[6] M. Andrews and L. Zhang. Scheduling algorithms for optimizing the tradeoffs between delay, queue size and energy. In *Proc. of IEEE CISS*, 2012.

[7] D. Ardagna, B. Panicucci, and M. Passacantando. A game theoretic formulation of the service provisioning problem in cloud systems. In *Proc. of WWW*, 2011.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud

computing. *Communications of the ACM*, 53(4):50–58, 2010.

[9] X.-R. Cao, H.-X. Shen, R. Milito, and P. Wirth. Internet pricing with a game theoretical approach: concepts and examples. *IEEE/ACM Trans. on Networking*, 10(2):208–216, 2002.

[10] J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In *Proc. of ACM HPDC*, pages 229–238, 2011.

[11] L. Columbus. Roundup of cloud computing forecasts and market estimates Q3 update, 2015.

[12] B. Darrow. AWS in fight of its life as customers like Dropbox ponder hybrid clouds and Google pricing. 2014. https://gigaom.com/2014/07/25/aws-in-fight-of-its-life-as-customers-like-dropbox-ponder-hybrid-clouds-and-google-pricing/.

[13] N. Devanur, J. Garg, R. Mehta, V. V. Vazirani, and S. Yazdanbod. A market for scheduling, with applications to cloud computing. *arXiv preprint arXiv:1511.08748*, 2015.

[14] Google Cloud Platform. Compute Engine Pricing. *https://cloud.google.com/compute/*, 2015.

[15] Google Cloud Platform. Preemptible Virtual Machines. *https://cloud.google.com/preemptible-vms/*, 2015.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62, 2009.

[17] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proc. of ACM IMC*, 2009.

[18] E. Kutanoglu and S. D. Wu. On combinatorial auction and lagrangean relaxation for distributed resource scheduling. *IIE transactions*, 31(9):813–826, 1999.

[19] Z. Liu, I. Liu, S. Low, and A. Wierman. Pricing data center demand response. In *Proc. of ACM SIGMETRICS*, 2014.

[20] R. Mahindra, H. Viswanathan, K. Sundaresan, M. Y. Arslan, and S. Rangarajan. A practical traffic management system for integrated LTE-WiFi networks. In *Proc. of ACM MobiCom*, 2014.

[21] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing cloud backend workloads: insights from Google compute clusters. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):34–41, 2010.

[22] C. Ng, D. C. Parkes, and M. Seltzer. Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. In *Proc. of ACM EC*, 2003.

[23] I. Paul. 10 alternative carriers that can save you serious cash on your smartphone bill. PC World, 2015. http://www.pcworld.com/article/2878298/10-alternative-carriers-that-can-save-you-serious-cash-on-your-smartphone-bill.html.

[24] D. Poola, K. Ramamohanarao, and R. Buyya. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Computer Science*, 29:523–533, 2014.

[25] C. M. Ramsay. Exact waiting time and queue size distributions for equilibrium M/G/1 queues with Pareto service. *Queueing Systems*, 57(4):147–155, 2007.

[26] 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. of ACM SoCC*, 2012.

[27] S. Ren, Y. He, and F. Xu. Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In *Proc. of IEEE ICDCS*, 2012.

[28] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. 2015.

[29] G. Sakellari and G. Loukas. A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39:92–103, 2013.

[30] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang. A survey of smart data pricing: Past proposals, current plans, and future trends. *ACM Computing Surveys (CSUR)*, 46(2):15, 2013.

[31] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das. Modeling and synthesizing task placement constraints in Google compute clusters. In *Proc. of ACM SoCC*, 2011.

[32] K. Song, Y. Yao, and L. Golubchik. Exploring the profit-reliability trade-off in Amazon's spot instance market: A better pricing mechanism. In *Proc. of IEEE/ACM IWQoS*, 2013.

[33] P. Upadhyaya, M. Balazinska, and D. Suciu. How to price shared optimizations in the cloud. *Proc. of the VLDB Endowment*, 2012.

[34] J. Wilkes and C. Reiss. ClusterData-2011-2. *https://github.com/google/cluster-data*, 2015.

[35] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, 2011.

[36] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely. Data centers power reduction: A two time scale approach for delay tolerant workloads. In *Proc. of IEEE INFOCOM*, 2012.

[37] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of EuroSys*, 2010.

[38] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE Trans. on Cloud Computing*, 2(1):14–28, 2014.

[39] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to bid the cloud. In *Proc. of ACM SIGCOMM*, 2015.

[40] Y. Zhou and D. Wentzlaff. The sharing architecture: sub-core configurability for IaaS clouds. In *Proc. of ACM ASPLOS*, pages 559–574, 2014.