Incentivizing Self-Capping to Increase Cloud Utilization

Mohammad Shahrad
Princeton University
mshahrad@princeton.edu

Cristian Klein
Umeå University
cklein@cs.umu.se

Liang Zheng
Princeton University
liangz@princeton.edu

Mung Chiang
Princeton University / Purdue University
chiangm@princeton.edu

Erik Elmroth
Umeå University
elmroth@cs.umu.se

David Wentzlaff
Princeton University
wentzlaf@princeton.edu

ABSTRACT

Cloud Infrastructure as a Service (IaaS) providers continually seek higher resource utilization to better amortize capital costs. Higher utilization not only can enable higher profit for IaaS providers but also provides a mechanism to raise energy efficiency; therefore creating greener cloud services. Unfortunately, achieving high utilization is difficult mainly due to infrastructure providers needing to maintain spare capacity to service demand fluctuations.

Graceful degradation is a self-adaptation technique originally designed for constructing robust services that survive resource shortages. Previous work has shown that graceful degradation can also be used to improve resource utilization in the cloud by absorbing demand fluctuations and reducing spare capacity. In this work, we build a system and pricing model that enables infrastructure providers to incentivize their tenants to use graceful degradation. By using graceful degradation with an appropriate pricing model, the infrastructure provider can realize higher resource utilization while simultaneously, its tenants can increase their profit. Our proposed solution is based on a hybrid model which guarantees both reserved and peak on-demand capacities over flexible periods. It also includes a global dynamic price pair for capacity which remains uniform during each tenant's Service Level Agreement (SLA) term.

We evaluate our scheme using simulations based on real-world traces and also implement a prototype using RUBiS on the Xen hypervisor as an end-to-end demonstration. Our analysis shows that the proposed scheme never hurts a tenant's net profit, but can improve it by as much as 93%. Simultaneously, it can also improve the effective utilization of contracts from 42% to as high as 99%.

CCS CONCEPTS

• Computer systems organization → Cloud computing; **• Software and its engineering → Cloud computing**; **• Social and professional topics → Pricing and resource allocation**; **Information system economics**; **• Theory of computation → Computational pricing and auctions**;

KEYWORDS

cloud computing, pricing model, economic incentives, utilization, resource management, dynamic pricing, SLA, IaaS

ACM Reference Format:
Mohammad Shahrad, Cristian Klein, Liang Zheng, Mung Chiang, Erik Elmroth, and David Wentzlaff. 2017. Incentivizing Self-Capping to Increase Cloud Utilization. In *Proceedings of ACM Symposium of Cloud Computing conference, Santa Clara, CA, USA, September 24–27, 2017 (SoCC'17)*, 14 pages. https://doi.org/110.1145/3127479.3128611

1 INTRODUCTION

Cloud computing promises to deliver computing and storage capacity at a usage-based price lower than self-hosting. By taking advantage of statistical multiplexing, cloud providers can host several cloud users, utilizing a capacity which is just a fraction of the sum of the cloud users' peak demands. This leads to higher infrastructure utilization and therefore lower costs [6].

Higher resource utilization can be a competitive advantage for IaaS providers, since they can amortize their capital as well as operational costs better to offer lower prices and/or achieve higher profit margins. Increasing server utilization is not only the best way to improve cost efficiency [9], but also an essential enabler of greener cloud services through better energy efficiency [20, 40]. Pushing for high infrastructure utilization, however, is rather arduous; mainly because cloud providers need to preserve a large spare capacity to manage demand fluctuations [1].

Solutions to this issue either involve more efficient provisioning of resources or new provisioning models that can offer inherently higher utilizations. Google's Borg [66] is an example of the former approach which employs techniques such as careful resource sharing and reclamation to improve utilization. In contrast, Amazon EC2's introduction of spot instances [4] was a successful new provisioning model which allowed selling unused resources with lower availability guarantees. Some other solutions include deploying long-term contracts [14], dynamic effective capacity modulation [68], and dynamic availability provisioning [54].

Graceful Degradation (GD) is a resilience concept widely used to enable IT services that can endure resource scarcity. One example of GD is that video quality can be reduced automatically when the network is slow so that the stream is not disrupted [61, 62]. Self-adaptation is also applied to cloud applications allowing them to survive temporary capacity shortages by degrading or disabling some of their features [33]. Researchers have already shown that using graceful degradation can improve the cloud resource utilization by 11 to 37 percentage points [63]. Moreover, it is easier to meet latency requirements for less bursty tenants [78].

In this work, we explore **how an Infrastructure Provider (IP) can give economic incentives to its tenants, the Service Providers (SPs), to use GD**. Furthermore, we investigate how the infrastructure provider can **incentivize them to use GD in the way it wants them to**; allowing the IP to achieve specific global policies. Our system supports GD-compliance and enables a mutually beneficial interaction by providing a pricing model for IPs and profit optimization means to their users. This results in less resource variation for IPs and more profit for users.

Our proposed solution is based on a hybrid model guaranteeing both reserved and peak on-demand capacities over flexible periods. It also includes a global dynamic price pair which remains uniform during each user's SLA term. Our paper supports making GD a first-class citizen in cloud-native applications and cloud provider APIs through the following main contributions:

- We propose a pricing model to incentivize GD-compliant SPs such that they can gain more profit by limiting their own burstiness. We provide formulations for SPs to select optimal reserved and peak on-demand capacity values assuming different price and revenue functions.
- We demonstrate how an IP can change global capacity prices to incentivize the same behavior among all its clients.
- Using simulations based on real-world service provider utilization traces, we evaluate our scheme's main promises regarding increased profit for SPs, improved effective utilization for IPs, and their ability to enforce global policies.
- We implement and test a prototype to validate the simulation results using RUBiS on the Xen [8] hypervisor.

The extent of our system's benefits depends on the revenue an SP makes from unit capacity and its sensitivity to GD. Our conservative analysis shows that while a tenant's profit is never hurt, using our pricing model can increase it by as much as 93%. It also can improve the effective utilization of contracts from 42% to as high as 99%.

2 BACKGROUND ON GRACEFUL DEGRADATION

Most cloud applications have rigid resource requirements, in the sense that, given a certain workload intensity — e.g., characterized by an arrival rate or a number of users — there is a fixed amount of computing, storage, and network capacity that the application requires to obtain a target performance — e.g., response time or throughput. If the application is not allocated the required capacity, it might overload or even thrash, affecting its delivered services. Conversely, if the application is allocated more than the required capacity, it cannot make effective use of it; hence capacity is effectively wasted.

In contrast, an application supporting GD can make effective use of a range of capacities. For each workload intensity and target performance, the application features a minimum and a maximum capacity. If the application is allocated less than the minimum required capacity, it is unable to deliver any useful service. If the application is allocated more than the minimum capacity, it can deliver useful service to its users, with the quality of experience increasing as more capacity is allocated to it. Beyond the maximum capacity, the application can no longer increase the delivered quality of experience, and the extra capacity is wasted.

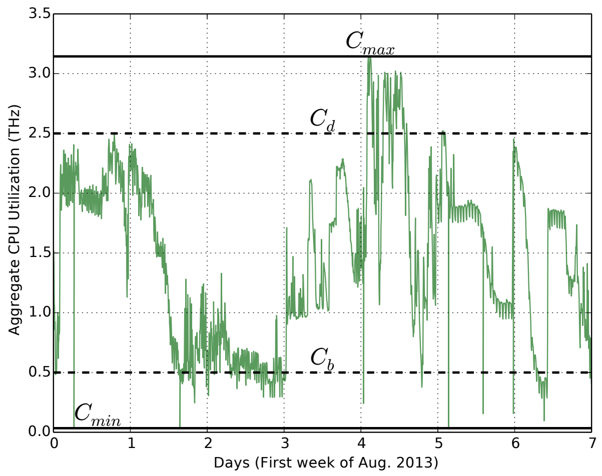

Figure 1: Aggregate CPU utilization variations for a service provider (Bitbrains [56]) serving 1,250 VMs.

Let us now examine two applications supporting GD, one for computing (CPU) capacity and the other for network capacity (bandwidth). Online shops generally offer end-users recommendations for similar products they might be interested in. No doubt, recommendation engines greatly increase the user experience, which translates to higher revenue. In fact, research has shown that recommendations may increase revenue by up to 50% [22]. However, due to their sophistication, such engines demand significant computing resources. By selectively activating or deactivating recommendations, an application's capacity requirements can be controlled at the expense of end-user experience. By applying a GD software engineering methodology, called brownout [33], the developer only needs to mark the recommendations as optional, and an external controller decides when to enable optional code, so as to maintain a given target response time (e.g. the 95th percentile response time below 300ms). At the minimum capacity, the online shop serves no recommendations, whereas at the maximum capacity it serves all users with recommendations.

As for network capacity, Dynamic Adaptive Streaming over HTTP [61, 62] is a technology to serve video streams at various levels of quality as permitted by network bandwidth. The video is divided into segments, usually 10 seconds, and each segment is encoded at several video resolutions, e.g., 240p up to 1080p. The video client continuously monitors the available bandwidth, by measuring how quickly the current segment was downloaded and decides the video quality of the next segment to download. From the service provider's perspective, if the minimum (network) capacity is allocated to it, all clients are served with lowest video quality; whereas if the maximum capacity is allocated to it, then all clients are potentially served with the best video quality.

3 SYSTEM OVERVIEW

An SP which is served on top of an IP's infrastructure can have capacity demand that varies and has unexpectedly high fluctuations over time due to variations in query rate, content, popularity, etc. As a result, SPs usually ask for more resources than their estimated peak capacity requirements to mitigate the impact of such

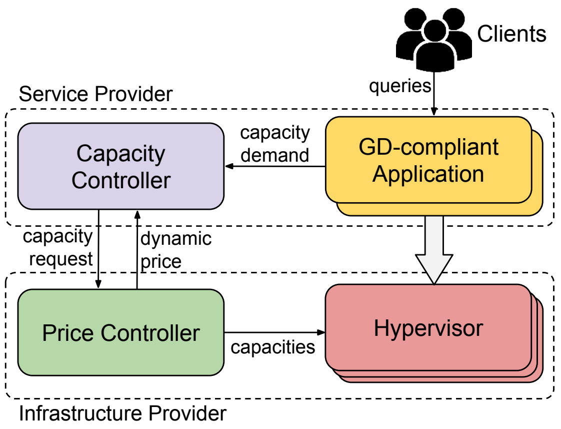

Figure 2: An overview of our system's architecture.

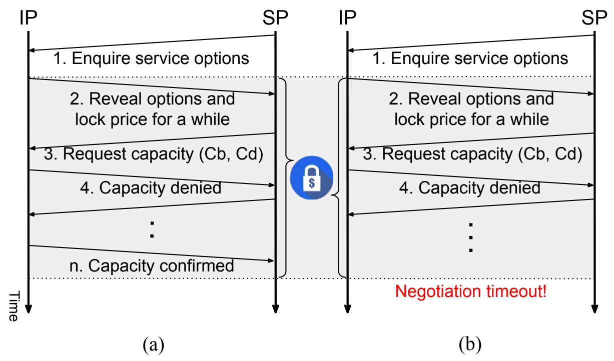
IP
SP
1. Enquire service options
2. Reveal options and lock price for a while
3. Request capacity (Cb, Cd)
4. Capacity denied
n. Capacity confirmed
Time
(a)
IP
SP
1. Enquire service options
2. Reveal options and lock price for a while
3. Request capacity (Cb, Cd)
4. Capacity denied
Negotiation timeout!
(b)

Figure 3: Negotiation process for a successful (a) and an unsuccessful (b) negotiation. The SP is offered guaranteed prices (p_b and p_d) for a fixed time window.

fluctuations on end-users' (clients') Quality of Service (QoS). We propose a new model for GD-compliant SPs, which includes a base reserved capacity c_b as well as total resource capacity c_d, as shown in Figure 1. The IP charges each SP a unit base price p_b for the reserved portion c_b, regardless of how much of the reserved capacity is going to be utilized. It also charges the SP a usage-based unit price p_d for any extra capacity usage between the base and the total capacity Service Level Objectives (SLOs) in a pay-as-you-go model. Any capacity request beyond c_d will not be allocated under the current SLA and requires re-negotiation.

An overview of our system is presented in Figure 2. Clients access the SP's application remotely. The application is assumed to support GD already, i.e., it gracefully degrades the quality of experience depending on the computing or networking capacity available to it. The SP has a capacity controller that negotiates the base capacity c_b and total capacity c_d with the IP, using algorithms provided later in the paper. The capacity requirement for serving all requests without GD is computed by the application itself, for example, as $c = \alpha\lambda$, where λ is a measure of load — such as number of users — and α is a constant obtained through off-line or on-line profiling representing the amount of load that a unit of capacity can sustain.

On the IP side, a price controller fulfills three roles. First, it gathers all capacities from each SP and computes the on-demand price (p_d). Second, to enforce total capacity and ensure that no SP uses more than what is allowed in the SLA, the price controller sets a capacity constraint for each SP. These capacity constraints are enforced by the underlying hypervisors running on the IP's host machines. Finally, the price controller meters the amount of base capacity requested and total capacity consumed by each SP over time for billing and planning purposes.

In Section 4, we present a pricing model which enables an SP to select the optimal resource capacity values based on given prices and enables an IP to control its overall utilization by setting the capacity prices. Dynamic pricing and graceful degradation are the two main elements that realize these objectives. The concept of dynamic resource pricing, which forms a feedback loop to control supply/demand mismatch as well as infrastructure under/over utilization, is not new and has been proposed previously by other researchers [29, 70, 74]. However, we provide SPs with GD as a tool to deal with this dynamic pricing environment more efficiently. Moreover, we believe that in order for dynamic pricing to be practical and easy-to-comprehend, the resource **price should stay constant during a tenant's term of contract**. Meaning that although the resource price might fluctuate continuously, as soon as a tenant confirms the SLA, it will be charged at a fixed price during its SLA period. We chose this for practicality and ease of use, but such an assumption is not fundamental in our proposed pricing model.

When service options are revealed to a tenant, prices are guaranteed for a short time window (e.g. an hour), within which the tenant can finalize the SLA. The IP might not approve the requested capacity due to limited available capacity or other reasons[1]. Therefore, finalizing a negotiation might take multiple iterations. The negotiation has to be restarted if the price freeze window is expired and SLA is not yet concluded. Figure 3 shows two example negotiation processes.

Defending against false-name bidding or collusion scenarios is an open issue for existing cloud pricing models [46], partially because performing large-scale collusion scenarios is rather infeasible and defending against them entails significant revenue reduction for cloud providers [69]. Similarly, we assume no such scenarios.

4 INCENTIVE-COMPATIBLE PRICING MECHANISM

We consider a cloud system, where a monopolistic IP allocates its resources of a total amount C to customers, including SPs who then deliver various cloud-based applications, such as online shopping and video streaming, to their end users. We suppose that some SPs can run GD-compliant applications, i.e., their user experience would be gracefully degraded if insufficient cloud resources are allocated to them (see Section 2). To incentivize such kind of SP self-adaption, we first describe the IP's pricing model in Section 4.1 and then discuss how SPs as well as the IP would benefit from our proposed scheme in Sections 4.2 and 4.3, respectively. Finally in Section 4.4 we discuss the statistical data required to use our scheme. Table 1 lists all parameters we use in this paper.

[1] Amazon EC2, for instance, sets default limits on its resources on a per-region basis [3].

Parameter	Description
β	Exponent of c in revenue function
c	An SP's aggregate capacity demand
$\hat{c}$	Delivered capacity to an SP
c_b	Reserved capacity portion
$c_b^\star$	Optimal c_b
c_d	Total deliverable capacity
$c_d^\star$	Optimal c_d
c_{max}	Maximum capacity demand in a period
c_{min}	Minimum capacity demand in a period
C	Total capacity of an IP
δ_d	Change in $c_d^\star$
δ_p	Change in p_d
$f(c)$	Capacity distribution function in a period
γ	Coefficient of power-law revenue function
k	Degree of homogeneity for revenue function
P	An SP's profit
p_b	Reserved capacity unit price
p_d	On-demand capacity unit price
$R(c, \theta)$	An SP's revenue function
θ	Service degradation factor
u_e	Effective resource utilization of an SP
Y	An SP's payment

Table 1: List of all parameters used in the paper.

4.1 Pricing Model

We described our model which consists of reserved (c_b) as well as total (c_d) capacity parameters in Section 3. While an SP is always charged for the reserved part (with unit price p_b), any extra capacity usage between the base and the total capacity SLOs is priced in a pay-as-you-go fashion (with unit price p_d). An SP (or any other IaaS customer in general) can trigger GD to deactivate or tune down some optional features to reduce the capacity.

Provisioning resources in a reserved manner is generally more favorable for IPs due to its lower risk. That is why such resources are usually provided with a lower price:

$$p_b < p_d. \tag{1}$$

It will be demonstrated later how these two pricing parameters affect the amount of reserved and total capacity an SP purchases. However, note that this simple equation works as a capacity valley shaping tool which motivates SPs to request as much reserved capacity as possible.

4.2 Service Provider: Trade-offs between Profit Maximization and Graceful Degradation

In Section 4.1, we mentioned that the SLA includes reserved (c_b) as well as total capacity (c_d) SLOs. Suppose that real-time capacity c for serving user demand of each SP follows a distribution with probability density function (PDF) $f(c)$, with a maximum capacity c_{max} and a minimum capacity c_{min}. We note that both c_b and c_d can be larger or less than c_{max} depending on an SPs' strategic decisions.

When c is larger than c_d, the degradation ratio for an SP capable of performing GD is

$$\theta = \frac{c_d}{c}, \quad (c_d < c), \tag{2}$$

meaning that in a balanced degradation, each user's delivered service or QoS (depending on the application) is roughly proportional to $\theta \in [0, 1]$. Notice that $\theta = 1$ always holds if $c_d \geq c_{max}$. For an SP with no GD capability, c_d must be at least c_{max} (no tolerance to capacity shortage) and θ is always 1 (no degradation).

Each SP's revenue function can be represented by $R(c, \theta)$, where c is the required resource assigned to serve all users with the premium service and θ is the percentage of c that is actually allocated to serve users. We make the following two assumptions for $R(c, \theta)$:

(i) Monotonically increasing in terms of c and θ:
$R(c, \theta_1) \geq R(c, \theta_2)$ if $\theta_1 \geq \theta_2$
$R(c_1, \theta) \geq R(c_2, \theta)$ if $c_1 \geq c_2$.
(ii) Positive homogeneity[2] of degree k in terms of θ:
$R(c, \lambda\theta) = \lambda^k R(c, \theta)$.

For the revenue function, the increasing monotonicity ensures the increase in revenue when serving more users. The positive homogeneous function has its well-known economic applications to model production functions [11, 31, 38], capturing the return of inputs (i.e. capacity) that scale up and down outputs (i.e. revenue). In particular, when the degradation ratio of θ increases by proportion λ, revenue increases by proportion λ^k.

For example, to model $R(c, \theta)$, we can use a general form of power-law functions:

$$R(c, \theta) = \gamma \theta^k c^\beta, \tag{3}$$

where γ is a positive constant representing the scale of the revenue to capacity demand, and $k > 0$ and $\beta > 0$ establish the revenue's power-law relation with θ and c. When $\gamma = 1/(1 - \alpha)$ and $k = \beta = 1 - \alpha$ for $\alpha \in [0, 1)$, (3) is the commonly-used α-fair function [48, 75].

Based on the Euler's homogeneous function theorem (cf. Theorem A.2, Appendix A), the degree of homogeneity k can be viewed as the ratio of marginal revenue to cost. For example, if $k \geq 1$, revenue increases more rapidly than the cost of cloud capacity does; thus, maximum profit for SPs happens when they claim a high enough $c_d = c_{max}$ to always guarantee the highest QoS for users. Conversely, the SPs with a diminishing marginal revenue (e.g., a concave revenue function, which is a special case for an increasing and positive homogeneous function of degree $k \in (0, 1)$) are more likely to take advantage of GD by setting $c_d < c_{max}$: whenever $c > c_d$, SPs would disable some of the client's alternative services, and the client's QoS is lowered as $\theta < 1$. This type of revenue function can be summarized as the following necessary condition (cf. Appendix A):

Proposition 4.1 (GD-profitable). *An application can increase its profit using GD if its revenue function $R(c, \theta)$ is positive homogeneous of degree $k \in (0, 1)$ in terms of θ. We call such an application GD-profitable.*

Figure 4 illustrates an SP's revenue and cost as a function of capacity demand (c), given some c_b and c_d values. When c exceeds c_d, SPs have to trigger GD to restrict their resources to c_d, leading to a diminishing revenue increase. While both cost and revenue increase with c, a GD-profitable SP can earn more profit by applying GD as long as marginal cost could be higher than revenue increase

[2]Definition and properties of positive homogeneity in Appendix A.

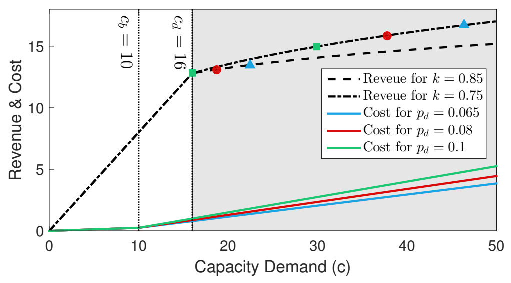

Figure 4: Illustration of revenue and cost for GD-profitable SPs. The revenue function is parametrized with $\gamma = 0.8, \beta = 1$. We suppose $c_b = 10$ and $c_d = 16$ and set $p_b = 0.025$. Gray area represents GD operation mode and colored markers show the profit maximization operating points for different k values and various cost functions.

when $c > c_d$: the markers on the revenue curves in Figure 4 show when maximum profit is achieved. However, under various revenue functions and pricing policies, c_b and c_d may not always maximize the SPs' profit for some capacity demands beyond c_d; for example, the marker on $c = c_d$ means that the maximum profit happens without GD. Thus, for SPs, c_b and c_d need to be carefully chosen based on p_b, p_d, and $R(c, \theta)$, and for the IP, it also needs to set p_b and p_d carefully to incentivize GD-profitable SPs.

We now mathematically formalize a GD-profitable SP's decision on its optimal capacity requests, $c_b^\star$ and $c_d^\star$.

Proposition 4.2. *If $p_b < p_d$ and the following two conditions are also satisfied:*

$$\int_{c_b}^{c_{max}} \frac{k}{c_b} R\left(c, \frac{c_b}{c}\right) f(c)\text{d}c > p_b, \tag{4}$$

$$\frac{k}{c_{max}} R(c_{max}, 1) < p_d, \tag{5}$$

then a GD-profitable SP maximizes its expected profit by choosing c_b and c_d such that $c_{max} > c_d > c_b > c_{min}$.

As a remark, we observe: the condition in (4) implies that reserved price is less than the expected unit revenue above c_b, while the condition in (5) ensures that the on-demand price is higher than the unit revenue at the peak capacity demand, i.e., SPs would not allow $c_{max} < c_d$. We thus find out that the GD-profitable SP is incentivized to reduce their peak capacity if conditions in (4) and (5) are satisfied.

Figure 1 illustrates the case in Proposition 4.2. If $c_{max} > c_d > c_b > c_{min}$, the expected payment for an SP can be calculated by

$$\mathbb{E}(Y) = p_b c_b + \int_{c_b}^{c_d} p_d(c - c_b) f(c)\text{d}c + \int_{c_d}^{c_{max}} p_d(c_d - c_b) f(c)\text{d}c, \tag{6}$$

and its expected revenue can be calculated by

$$\mathbb{E}(R) = \int_{c_{min}}^{c_d} R(c, 1) f(c)\text{d}c + \int_{c_d}^{c_{max}} R(c, c_d/c) f(c)\text{d}c, \tag{7}$$

leading to the expected profit of

$$\mathbb{E}(P) = \mathbb{E}(R) - \mathbb{E}(Y). \tag{8}$$

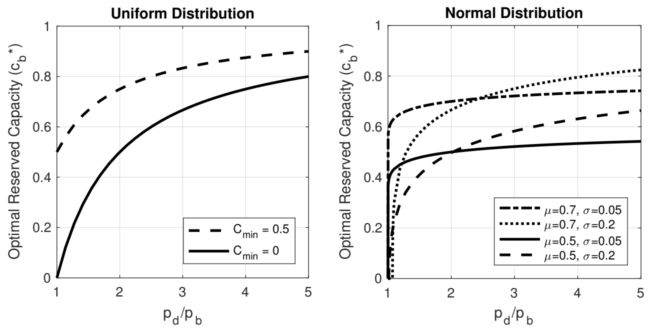

Figure 5: The optimal reserved capacity (c_b^*) based on Corollary 4.3 for different demand distribution ($f(c)$) functions. c_{max} is normalized to one for all of them.

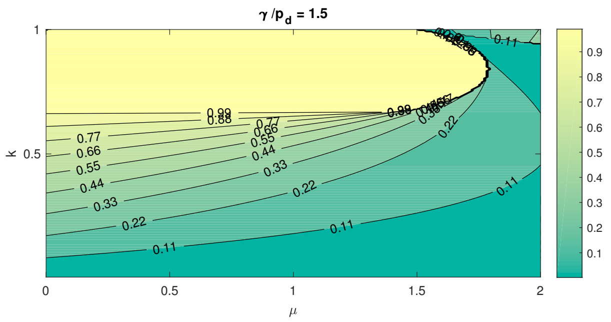

Figure 6: The optimal total capacity (c_d^*) based on (10) for a power-law (Eq. (3)) revenue function under a uniform demand distribution. C_{max} is normalized to one.

Corollary 4.3. *In order to maximize the expected profit given in (8), the optimal $c_b^\star$ for any SP and the optimal $c_d^\star$ for a GD-profitable SP satisfy*

$$\int_{c_b^\star}^{c_{max}} f(c)\text{d}c = \frac{p_b}{p_d}, \tag{9}$$

$$c_d^\star = \frac{\int_{c_d^\star}^{c_{max}} kR(c, c_d^\star/c) f(c)\text{d}c}{\int_{c_d^\star}^{c_{max}} p_d f(c)\text{d}c}. \tag{10}$$

This corollary, the proof of which is provided in the Appendix B, allows SPs to find the optimal base and total capacities to maximize their profit. Note that all SPs can choose $c_b^\star$ satisfying (9) regardless of their GD-profitability. GD-profitable SPs set $c_d^\star$ based on (10) and if $c_b^\star$ was larger than $c_d^\star$, limit it by $c_d^\star$. In contrast, GD-unprofitable SPs must set $c_d^\star$ to c_{max}, since they cannot tolerate any capacity shortage.

We can observe from (9) that $c_b^\star$ is inversely related to $\frac{p_b}{p_d}$, i.e., a smaller p_b encourages SPs to reserve more cloud resources. Meanwhile, (10) implies p_d is greater than $\int_{c_d^\star}^{c_{max}} \frac{k}{c_d^\star} R(c, c_d^\star/c) f(c)\text{d}c$, verifying the intuition of the relationship between marginal cost and marginal revenue. Figure 5 shows $c_b^\star$ as a function of the price ratio and for some different demand distribution ($f(c)$) functions. Figure 6 depicts $c_d^\star$ for different revenue function variables.

4.3 Infrastructure Provider: Controlling Resource Utilization with Price

Let us define the effective utilization of each SP to be the ratio of utilized resources ($\hat{c}$) to the requested total capacity (c_d):

$$u_e(t) = \frac{\hat{c}(t)}{c_d} = \frac{min(c(t), c_d)}{c_d}. \qquad (11)$$

Here, $\hat{c}(t)$ is the **provisioned capacity** which is bounded by c_d, whereas $c(t)$ is the **capacity demand** which might not be fully granted. When GD-profitable SPs are incentivized to degrade service (limit $c(t)$ by c_d), their effective utilization rate is improved. This is achieved essentially by lowering the peak-to-average ratio of the capacity usage. Improved effective utilization benefits the IP by decreasing spare capacity that was required to provision infrequent fluctuations. Such reclaimed resources can either be re-sold to increase revenue, or put into the low-energy mode to decrease cost for the IP.

From (9) and (10), recall that the optimal amounts of reserved ($c_b^\star$) and total capacity ($c_d^\star$) requested by SPs are functions of their price, p_b and p_d. This enables the IP to control the resource utilization by shaping SPs' behavior on resource requests. Therefore, we characterize $c_b^\star$ and $c_d^\star$ in (9) and (10) with regard to p_b and p_d:

PROPOSITION 4.4. *The optimal reserved capacity $c_b^\star$, as given in (9), has a direct relationship with $\frac{p_d}{p_b}$, while the optimal total capacity $c_d^\star$, as given in (10), has a inverse relationship with p_d.*

As we can observe from (11), the key to the IP's utilization rate lies in the value of $c_d^\star$. Leveraging Proposition 4.4, we further quantify this monotonic dependency between $c_d^\star$ and p_d in the following.

COROLLARY 4.5. *Suppose $k \in (0, 1)$. When p_d changes by $\delta_p \in (0, 1)$, i.e., $\tilde{p}_d = (1 \pm \delta_p)p_d$, then the optimal total capacity $c_d^\star$ decreases/increases by $\delta_d \in (0, 1)$, i.e., $\tilde{c}_d^\star = (1 \mp \delta_d)c_d^\star$, where $\delta_d \geq \left|1 - (1 \pm \delta_p)^{-\frac{1}{1-k}}\right|$.*

The above proposition and corollary, proof of which can be found in Appendix C, address the monotonic dependence of controlled variables (optimal capacities) on system inputs (capacity price). Although how k affects SPs' revenue remains unknown to the IP, we find that the lower bounds $1 - (1 + \delta_p)^{-\frac{1}{1-k}}$ and $(1 - \delta_p)^{-\frac{1}{1-k}} - 1$ of the changes in the total capacity $c_d^\star$ due to both positive and negative changes in p_d are increasing functions of $k \in (0, 1)$. Thus, Corollay 4.5 can be further relaxed: an increasing/decreasing change of the price p_d by δ_p must lead to a decreasing/increasing change of δ_d in $c_d^\star$ that satisfies $\delta_d \geq \delta_p/(1 \pm \delta_p)$. Such a dependence is ideal for control loops and can empower a robust feedback mechanism.

The significance of this proposition is that it holds for all GD-profitable SPs simultaneously. Therefore, **changing the global capacity price will incentivize the same degradation behavior among all GD-profitable SPs**. At the same time, for GD-unprofitable SPs, price variations translate into a supply demand control mechanism existing in current dynamic markets that do not support GD (e.g. spot instances [4]). Table 2 presents how global reserved and on-demand prices should change to accomplish certain objectives.

Desired objective	Required capacity	Global price
Increase utilization	($c_b \uparrow$,$c_d \uparrow$)	($\frac{p_d}{p_b} \uparrow$,$p_d \downarrow$)
Increase effective utilization (Decrease Footprint [30])	($c_b \uparrow$,$c_d \downarrow$)	($\frac{p_d}{p_b} \uparrow$,$p_d \uparrow$)

Table 2: Proposed monolithic incentive mechanism enables the IP to accomplish its objectives through global pricing.

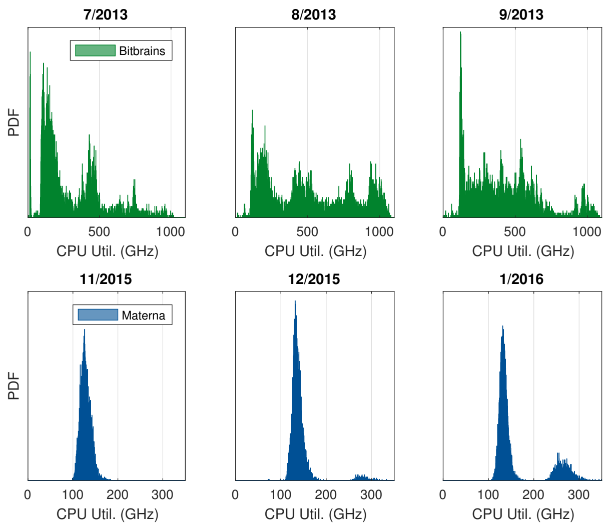

Figure 7: Varying probability density function (PDF) for aggregate CPU utilization of two service providers over three months.

Definition 4.6 (GD-compliant). If a GD-profitable SP is capable of performing GD to achieve $c < c_{max}$, it is GD-compliant. A GD-noncompliant SP is a GD-profitable SP that is not GD-compliant.

GD-compliance means that the SP has technically implemented resilience to capacity shortage, e.g. by serving some product pages without recommendations, and it makes financial sense for the SP to trim its peak demand, e.g., deactivate recommendations during peak hours to reduce infrastructure cost. In what follows, we focus on GD-profitable SPs and discuss the benefits that the GD-compliance brings.

4.4 Determining Resource Distribution

Knowing the resource distribution function ($f(c)$) is required to determine the optimal capacities using relations (9) and (10). The more knowledge an SP has on its future resource distribution, the more precisely it can decide c_b and c_d values. In this section, we discuss how an SP's distribution might vary and how it can predict its future distribution. Later in Section 5, we evaluate the impact of imperfect predictions on the benefits of our proposed scheme.

Analyzing real world traces, we observed that resource distributions could vary considerably depending on workloads and might not follow typically known trends (e.g. a normal distribution). For instance, Figure 7 shows how such monthly distribution functions changed for two service providers, Bitbrains [56] and Materna [34, 35], during three consecutive months. Although both

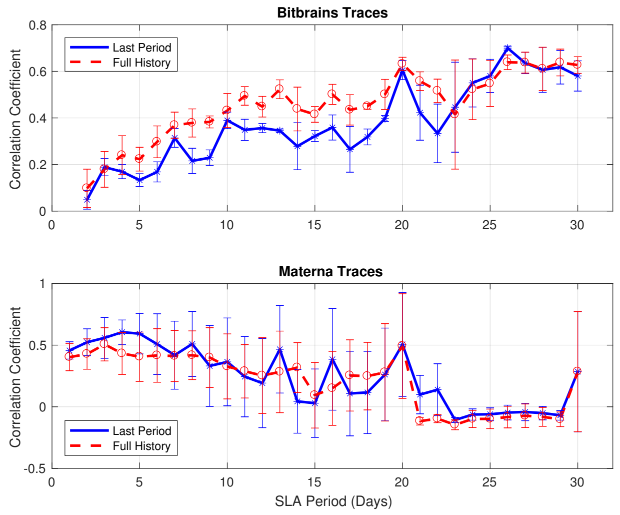

Figure 8: Prediction accuracy for capacity distribution based on last period and full history as a function of period length for different traces.

providers serve business-critical applications for enterprise customers, distributions of their CPU utilization are quite different. However, the resource distribution of each SLA period has some observable similarity with the previous periods. Such similarity can be used for predicting future distributions.

These two workload traces are specifically interesting for our study since they have inherently distinct characteristics which make their predictability very different. Figure 8 shows the correlation coefficient, a measure of similarity [50], of resource distributions between each period with either its previous period or the entire observed history. While for the Bitbrains traces, using a longer period and considering full history generally leads to a more precise prediction measure, having shorter periods and only considering the most recent period can work better for Materna traces. Furthermore, a service provider can use its resource usage history to determine the optimal SLA period which maximizes its resource predictability.

5 NUMERICAL EVALUATION OF INCENTIVES

In this section, we use numerical simulations to better characterize our proposed scheme. These numerical analyses allow fast design space exploration and are complementary to the actual evaluation of the implemented prototype in Section 6. We use the Bitbrains [56] performance traces in this section. Results are attained under fixed resource price (p_b = \$0.05 and p_d = \$0.07 per VM-Hour [2]) throughout the simulation unless otherwise specified.

Figure 9 shows submitted capacities from two service providers, one of which is GD-compliant. Both SPs have weekly SLA periods and have the same revenue function of

$$R(c, \theta) = 0.08 \times \theta^{0.7} c \text{ (\$ per VM-Hour)}, \tag{12}$$

where c is the total CPU capacity demand in VM unit (assumed 2926 MHz for a VM) and θ is the degradation factor (see Section 4.2 for details). Both SPs use a period's capacity demand distribution as the predictor for the next period. While both SPs can select the

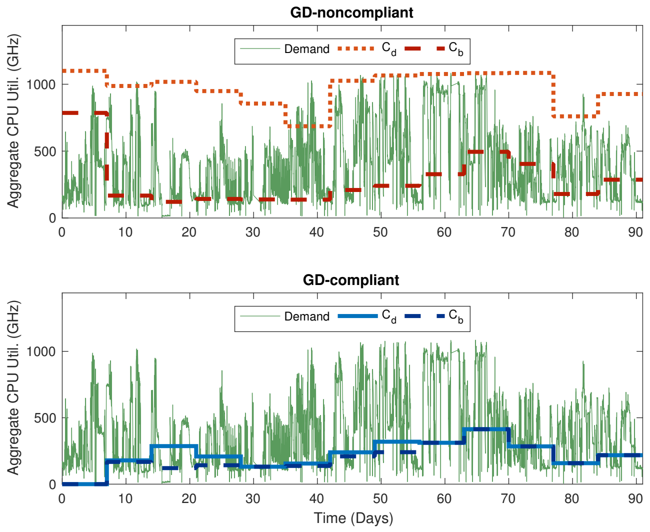

Figure 9: Requested capacities for a GD-noncompliant and a GD-complaint SP. They have similar price and revenue functions and run under weekly SLA periods.

optimal reserved capacity, $c_b^\star$, using (9), the GD-compliant SP can also select optimal total capacity ($c_d^\star$) values less than c_{max}. In contrast, the GD-noncompliant SP needs to avoid capacity shortage ($c_d^\star$ is set to c_{max} from the previous period), implying higher costs, but observes no degradation ($\theta = 1$), implying higher revenue. In the rest of this section, we will show how such GD can improve an SP's profit while enhancing the effective utilization of resources.

5.1 Increased Service Provider Profit

GD-compliance can improve profit of SPs by two inherently similar mechanisms. First, by diminishing the negative influence of unexpected bursts on the revenue; and second, by deliberately neglecting known occasional bursts that are costly to serve.

Figure 10 depicts the profit of two service providers in a 3-month window as a function of their SLA period length. While one of the SPs is GD-compliant and optimizes both capacity parameters, the GD-noncompliant SP can only optimize the reserved capacity (c_b). Their revenue functions are the same as (12). We consider cases where the future demand distribution is either predicted simply by only observing the previous period (see Section 4.4) or is fully known (oracle). The latter is used as the upper limit for prediction quality. Here are our observations from Figure 10:

- GD-compliance with the simple prediction mechanism can improve profit by 15.8% on average (28.5% maximum). Likewise, GD-compliance with the perfect prediction provides an average profit improvement of 11.2% (18.6% maximum). A better prediction of $f(c)$ can help all SPs, regardless of their GD-compliance, gain more profit by choosing $c_b^\star$ more precisely.
- Better prediction quality can improve the profit significantly. A perfect prediction can offer, on average, 7.1% and 11.5% higher profit for GD-compliant and GD-noncompliant SPs, respectively.

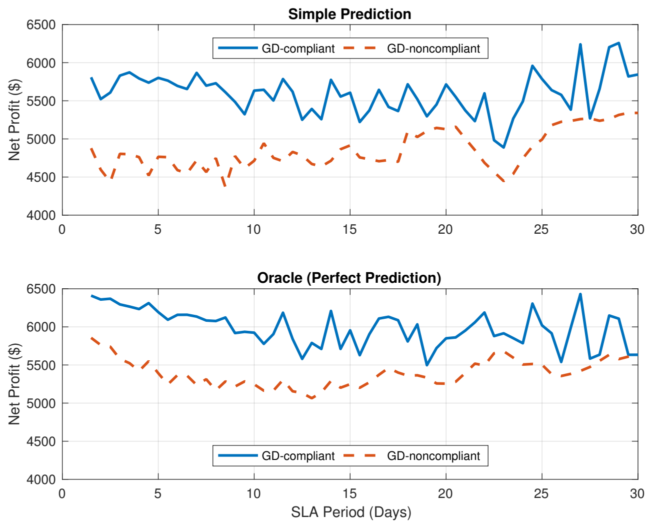

Figure 10: GD-compliance increases profit regardless of the SLA period length.

- Profit increase is generally higher for shorter SLA periods. The demand distribution function ($f(c)$) used by our optimizers varies over time. The shorter the period, the more fit the optimized capacities would be to all demand values over the period.

5.2 Increased Effective Utilization

We introduced the effective utilization, u_e, in (11), which is a metric indicating how much a user has utilized its requested capacity. One of the main promises of our proposed scheme is to improve the effective utilization via pricing incentives for GD-compliant service providers. Figure 11 shows the average effective utilization of the two service providers discussed in the previous subsection, using previously mentioned capacity distribution prediction methods. Some of our main observations in this figure are as follow:

- As seen in Figure 11, GD-compliance can improve the effective utilization noticeably; on average from 41% to more than 73% for simple prediction and from 43% to 78% for the oracle run.
- GD-compliance with a primary predictor can achieve 88.9% effective utilization with a two-day period length. Such 1.8x improvement is accompanied with a 26.4% profit increase for GD-compatible SP (Figure 10).
- Effective utilization is generally better (higher mean and less variance) for shorter period lengths. As mentioned earlier, using longer period usually translates into a broader demand variation range, which makes simultaneous optimization of profit and effective utilization less efficient.

The degree of improvement in effective utilization depends on how motivated an SP is to perform GD. Motivation to use GD depends on two factors; the $\frac{Revenue}{Cost}$ ratio and sensitivity to degradation. While the $\frac{\gamma}{p_d}$ ratio is a good representative for the former, parameter k models the latter (lower k means less sensitivity). Figure 12 shows how these two impact the average u_e improvement. We use a weekly SLA period and other parameters are similar to the previous analysis. High effective utilization is achieved through GD if p_d is high compared to the revenue. However, if revenue is much larger than the highest capacity price, p_d, SP has no incentive to apply GD and u_e remains unchanged. At the same time, less sensitivity to degradation hurts an SP's revenue less, effectively leading to the same trade off.

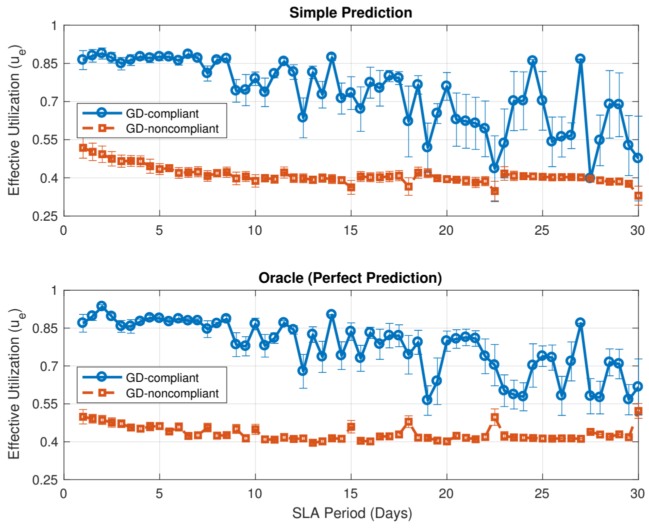

Figure 11: The average effective utilization of GD-compliant customers increases significantly regardless of their period length.

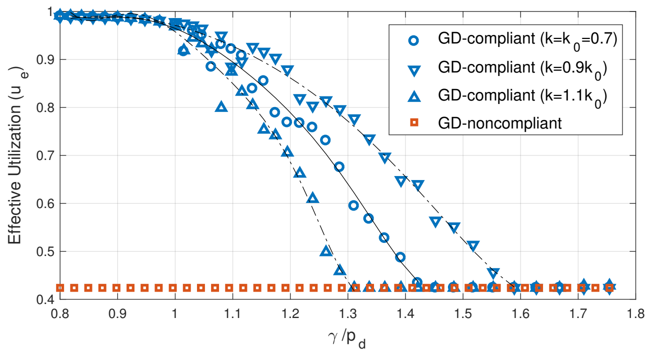

Figure 12: Effective utilization improvement for an SP depends on how motivated it is to practice GD. Profitability of GD is modeled by the $\frac{\gamma}{p_d}$ ratio as well as the k parameter (sensitivity to degradation).

5.3 Multi-SP Dynamics

To show a more complex dynamic when serving multiple SPs, we have selected three subsets of the Bitbrains performance traces and treated them as separate SPs. The number three is for the sake of easy presentation in this paper. All SPs have 3-day SLA periods with their period initiation one day apart from each other. They update their optimal c_b and c_d at the start of each period with respect to current IP prices. While p_b is constant, the IP dynamically sets p_d on a daily basis to encourage/discourage GD. Figure 13 shows the resource consumption of those SPs as well as variations of $\frac{\gamma}{p_d}$ due

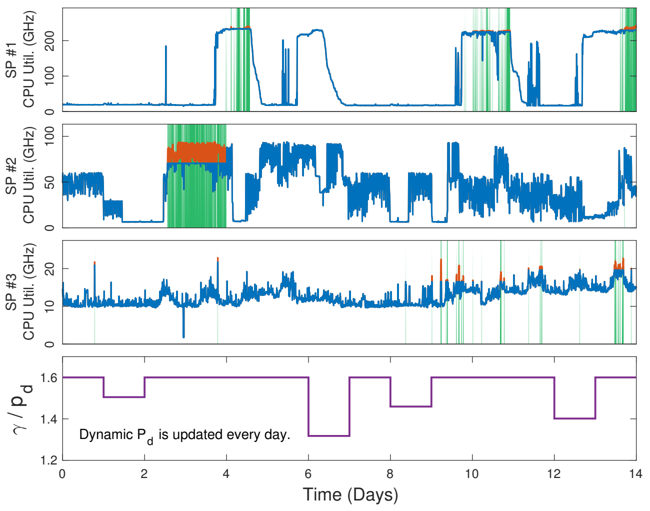

Figure 13: Resource consumption of three GD-compliant SPs over two weeks. While p_b is constant, the IP dynamically sets p_d on a daily basis to encourage/discourage GD. Degradation occurrences are highlighted by green vertical bars.

$\frac{p_d}{p_b}$	GD-noncom.		GD-comp.		Profit Increase (%)		Avg. Effective. Util	
	k	γ/p_d	k	γ/p_d	Predict	Oracle	Predict	Oracle
1	0.5	1.14	0.5	1.14	93.3	92.5	(42,85)	(43,86)
	0.7	1.14	0.7	1.14	29.2	32.7	(42,87)	(43,88)
	1	1.14	1	1.14	0.0	0.0	(42,42)	(43,43)
	0.5	1.5	0.5	1.5	7.2	7.0	(42,80)	(43,81)
	0.7	1.5	0.7	1.5	0.0	0.0	(42,43)	(43,43)
	1	1.5	1	1.5	0.0	0.0	(42,43)	(43,43)
1.4	0.5	1.14	0.5	1.14	54.4	45.9	(42,85)	(43,86)
	0.7	1.14	0.7	1.14	24.3	17.1	(42,87)	(43,88)
	1	1.14	1	1.14	0.0	0.0	(42,43)	(43,43)
	0.5	1.5	0.5	1.5	7.2	7.0	(42,80)	(43,81)
	0.7	1.5	0.7	1.5	0.0	0.0	(42,42)	(43,43)
	1	1.5	1	1.5	0.0	0.0	(42,42)	(43,43)

Table 3: The impact of change in the capacity price and revenue function parameters on profit gains. (SLA period = 1 week, $\beta = 1$).

to p_d changes. Degradation occurrences are highlighted by green vertical bars. Although these SPs utilize different capacity ranges, they all experience GD at sudden burst periods, especially after price increase ($\frac{\gamma}{p_d}$ decrease). As discussed in Section 4.3, an IP can use the pricing parameters to control resource utilization. While we have introduced strong guarantees such as Corollary 4.5, we leave designing control mechanisms atop of them for future work.

5.4 Sensitivity Study

Our pricing scheme consists of several parameters. In this section, we describe their relationship intuitively, discuss what are reasonable ranges for them, and evaluate how sensitive the results are to changes in those parameters. Instead of presenting the multidimensional search space to readers, we include some limited cases in Table 3.

We first elaborate that certain price and revenue ratios determine the requested capacities. As seen in Section 4, the $\frac{p_d}{p_b}$ ratio affects the selection of optimal reserved capacity ($c_b^\star$) and the ratio of $\frac{\gamma}{p_d}$ influences the optimal total capacity ($c_d^\star$). This means that if p_b, p_b, and γ (which corresponds to the revenue function) are all scaled by the same factor, optimal capacity values are unchanged, leading to a profit scaled by the same factor. Therefore, none of the profit improvement or effective utilization enhancement results vary with such a scaling. That is why we included only the ratios in Table 3.

A major takeaway from Table 3 is that if an SP is making too much revenue from the unit capacity compared to the price it pays for it (high $\frac{\gamma}{p_d}$ ratio), it does not make any economic sense to consider going into the GD mode to save money. However, the closer revenue becomes to the capacity price and the less sensitive the delivered service is to degradation (lower k), the higher is the incentive for an SP to employ GD. While the table shows that profit gain can be as high as 93.3% for these parameters, we would rather emphasize the fact that SPs never lose money using GD. Also, a smaller $\frac{p_d}{p_b}$ ratio leads to smaller reserved capacity. So the other takeaway from this table is that the lower the optimal reserved capacity, the more our scheme has to offer.

6 PROTOTYPE EVALUATION

To demonstrate the feasibility of our approach in a practical system and validate the results obtained through numerical analysis, we report results obtained using our implementation of economic incentives for graceful degradation. In what follows, we first describe our experimental setup and then show how the experiments validate results obtained numerically. Eventually, we test the scalability of our implementation by stressing the contention point our system.

6.1 Experimental Setup

Experiments were conducted on a single physical machine equipped with two AMD Opteron™ 6272 processors[3] and 56 GB of memory. We use Xen [8] as the hypervisor since, to the best of our knowledge, it is the only hypervisor that supports vertical CPU scaling for Virtual Machines (VMs) [39]. For example, allocating 800% CPU means that the application had exclusive access to 8 cores of the physical machine, while 50% signifies accessing a single core of the physical machine, for half of the time. Combined multiplexing of the physical cores, both in space and in time, is common in today's virtualized data-centers [25]. Furthermore, the fact that Amazon, the front-runner in spot instances and micro instances, runs a modified version of Xen is also an indication of its versatility and CPU scheduling capabilities.

Our SP deploys a single VM, which runs RUBiS, an eBay-like e-commerce prototype, that is widely-used for cloud benchmarking [16, 24, 57, 59, 60, 65, 77]. RUBiS already supports graceful degradation, thanks to a previous contribution [33]. The number of requests served with recommendations is modulated so as to maintain a 95th percentile response time of 1 second.

We made sure that the results are reliable and unbiased as follows: We used the `vmtouch` utility [26] to hold the database files in-memory, thus avoiding variance due to hard disk latency; to

[3]2100 MHz, 16 cores per processor, no hyper-threading.

ensure the load is generated in the same way during each experiment, we used the `httpmon` [32] workload generator in open system model [51] and the same sequence of exponentially distributed inter-arrival times; no non-essential processes or `cron` scripts were running at the time of experiments.

To foster replication of our results [21, 67] and make our contribution more useful to other researchers, we published our experimental setup, which includes "infrastructure as code" as Ansible scripts, under an open-source license.[4]

6.2 Validation of Numerical Results

To test the behavior of our implementation, we ran the following experiment. We took ten consecutive days of the BitBrains traces and scaled them down in two dimensions: First, time-wise, we compressed the traces by a factor of 60, i.e., the five-minute measurement period in the original trace became five seconds in our experiment. This allowed us to get useful results within 4 hours while giving the application enough time to adapt to the necessary degradation level between consecutive load levels. Second, magnitude-wise, we scaled the capacity demands by a factor of 2×10^{-5}, so that the load fits within the 30 cores of our testbed. The resulting load trace was used as an input for time-varying average arrival rate to our workload generator. The workload generator uses a Poisson process to generate the actual arrival time of each request; hence the observed arrival rate features a large variance around the average given as input. **Note, however, that the demand generated by the workload generator is unavailable to the GD application, which, instead, must measure it.**

The implementation was configured as follows. We used the same p_b and p_d as in Section 5, and a revenue function $R'(c, \theta) = 2 \times 10^{-5} \times R(c, \theta)$ (cf. Eq. (12)). The SLA terms are 24 hours long, i.e., 24 minutes experiment time. To ensure that VMs can gather correct measurements, the SP enforces $c_d \geq c_b \geq 1$.

Figure 14 presents the experimental results. During the first SLA term, the SP had no previous knowledge on its distribution of demand, so it requested $c_b = c_d = 1$. This was enough capacity for it to cope with the incoming arrival rate, albeit degraded between 75% and 100%, and learn the demand distribution for future predictions.

During the second term, the previously learned demand distribution was used to compute suitable c_b and c_d values. The requested capacities were enough to cope with the load with negligible degradation. Next, during the third term, c_d was slightly reduced since the peaks in the second term were smaller than those encountered in the first term. At the same time, c_b was slightly increased, since the load rarely went too low, hence cost saving could be achieved by increasing the base reserved capacity, which is cheaper than dynamically requested capacity. Some peaks were encountered, which were coped with using GD. Indeed, thanks to our contribution, the SP is incentivized to activate GD when encountering a peak instead of over-provisioning.

Similar observations can be drawn for the other SLA terms. The chosen c_b and c_d seem delayed by one SLA term when compared to the demand. This is due to the fact that our implementation cannot use an oracle to know future demand and must instead rely on predictions based on the demand in the previous SLA term.

[4]https://github.com/cristiklein/gdinc-experiment

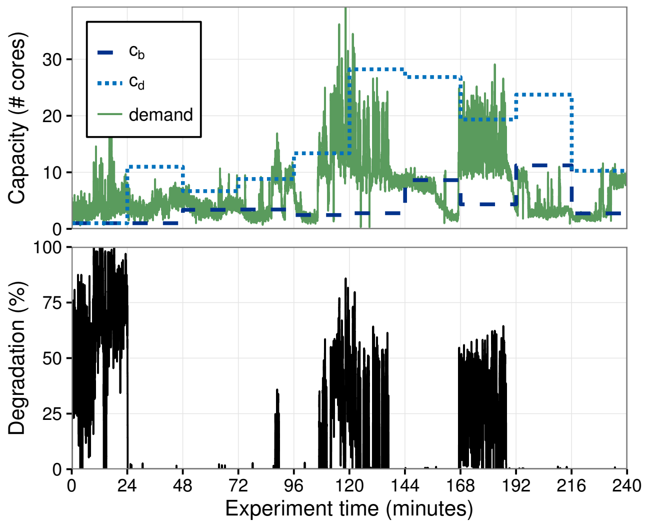

Figure 14: Validation of numerical results through a real implementation using a GD-compliant SP. The x-axis represents experiment time after compressing the Bitbrains trace. SLA renegotiations are highlighted with vertical grids. Degradation represents the fraction of requests that the SP had to serve without recommendations to maintain the 95th percentile response time below 1 second. The requested c_b and c_d closely follow the demand, with one SLA term lag, but trim the peaks, as the SP is incentivized to activate GD.

6.3 Scalability

For testing the scalability of our approach, we focus on the price controller (see Fig. 2), which is the contention point of our approach; the other components feature one instance either per physical machine (hypervisor) or virtual machine (capacity controller). Given our limited experimental testbed, we used "stub SPs" which are only composed of the capacity controller with simulated application load, but have no actual application to run.

We tested the scalability of the price controller up to 10,000 SPs, which showed a linear increase of the duration of its control loop as a function of the number of SPs in the system. Even with 10,000 SPs the duration was only 0.457 seconds, which means that the IP can quickly adjust the prices in case data-center utilization is getting too low or too high levels. In case the data-center needs to support more SPs, scalability can be increased by partitioning the data-center and assigning one price controller per partition, with SPs being assigned a partition as they enter the system.

On the SP's side, the instantaneous capacity requirement for serving all requests without GD is computed based on a method that is previously developed [36, 37]. The whole implementation has constant complexity and only requires a few floating-point computations every time a new value for capacity is computed.

7 RELATED WORK

1) Increasing Resource Utilization. High infrastructure utilization is critical to maximize the return of investment [9] and energy efficiency [10]. Within a single application, this can be achieved by careful resource provisioning to reach a given performance goal [17, 49, 71] or reducing the required headroom [41, 44]. At the infrastructure level, mapping algorithms are used to co-locate applications with antagonist resource requirements [13, 43, 66]. Since most IaaS tenants tend to overprovision their VM demands, resource overbooking (over-commit) [23, 63, 64] was used to accept more tenants if the resource usage is predicted to allow so without SLA violations. Performance-based service differentiation [36] may be used to ensure "gold" applications always maintain their target performance, while "bronze" ones are degraded if actual usage was mispredicted.

Several other provisioning models have been proposed to improve resource utilization. Capacity modulation [68] can further increase utilization by making both pricing and performance of VMs volatile. Providing long-term SLOs [14] is another way to enhance the value of reclaimed resources. Availability Knob [54] has been proposed to provide a variety of availability guarantees, improving utilization of reliability-heterogeneous infrastructures. Similar to our work, Morpheus [30] has exploited lowered performance variance to improve cluster utilization, but through automated SLOs as opposed to market incentives. Finally, many solutions have been proposed at the architecture-level to enable better utilization of underlying cloud processors [7, 47].

2) Self-Adaptation and Graceful Degradation. Self-adaptation is a software engineering method to reduce runtime uncertainty, by allowing an application to adapt to internal or external dynamics [18]. GD can be seen as a self-adaptation feature to maintain a given QoS goal — e.g., no video lag, low response time — despite uncertainty in the available amount of computing or networking capacity. Such adaptation is particularly important in multi-tenant environments, such as cloud computing, which feature the "noisy neighbor" phenomenon [12].

Cloud applications can support GD through brownout [33]: parts of the response are marked as optional, and a controller decides when to enable the optional code. With proper coordination between an SP and its IP, brownout can be used to compensate for overbooking [63]. In this work, we tackle the real deployment of such methods by incentivizing tenants to adopt them.

3) Pricing to Shape Demand and Behavior. Dynamic pricing is an effective mechanism to stabilize demands for networking, computing, and utility resources [28, 52, 53]. Auction-based pricing [55, 76] has been introduced in IaaS cloud markets to encourage SPs' consuming spare resources, where they bid against dynamically-set cheaper spot prices with no or little availability guarantees, e.g., Amazon EC2's *spot instances* [4, 42].

Many works considered the incentivizing problem from an operational perspective, but their primary objectives are either job scheduling [27, 72] or IP revenue maximization [58, 70]. Game theory principles can be applied to build incentive compatible pricing models that enforce mutually truthful behaviors in cloud markets [54]. Although Chaisiri et al. [15] suggested leveraging on-demand and reserved prices to cope with demand uncertainty, cloud resource provisioning is modeled as a dynamic program without considering interactions between the IP and SPs.

4) Workload Prediction. Workloads are generally predicted with short horizon for predictive auto-scalers [45] or with long horizon for dimensioning of physical infrastructure [19]. For our setting, predicting PDF of demand is sufficient, as opposed to the exact demand function. We did not introduce new prediction mechanisms as we believe current solutions (e.g. Cyclic Window Learning Algorithm [73]) are sufficient for our scheme. On the contrary, we demonstrated substantial benefits of our scheme using a simple predictor and compared the result against the perfect prediction.

8 CONCLUSION

Achieving high resource utilization is difficult due to IPs needing to maintain spare capacity to service demand fluctuations. Previous work has shown that graceful degradation, a technique originally designed for constructing robust services, can be used to improve resource utilization in the cloud by absorbing such demand fluctuations. In this work, we proposed a scheme that enables IPs to give economic incentives to their tenants to use graceful degradation. We evaluated our scheme using both simulations and implementing a prototype and showed that while it never hurts a tenant's net profit, it can improve it by as much as 93%. Simultaneously, it can also improve the effective utilization of contracts from 42% to as high as 99%. We tie profit maximization of tenants to higher utilization of claimed resources, through which we create a mutually beneficial model that can lead to greener cloud services.

ACKNOWLEDGMENTS

We thank Jiasi Chen, Carlee Joe-Wong, Alexey Lavrov, Fariborz Salehi, anonymous reviewers, and our paper's shepherd, Siddhartha Sen for their feedback on this work. This work was partially supported by the NSF under Grants No. CCF-1453112, CCF-1438980, CNS-1347234, CNS-1456847, AFOSR under Grant No. FA9550-14-1-0148, DARPA under Grant No. N66001-14-1-4040, ARO under Grant No. W911NF14-1-0190, and the Swedish Wallenberg Autonomous Systems and Software Program (WASP). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

APPENDIX

A POSITIVE HOMOGENEOUS FUNCTION

Definition A.1. A continuously differentiable function $h: \mathbb{R}^+ \to \mathbb{R}$ is positive homogeneous of degree k if

$$h(\lambda x) = \lambda^k h(x). \tag{13}$$

We point out a useful property of positive homogeneous functions. Although they can also be found in many textbooks, we state them here for mathematical completion.

THEOREM A.2 (EULER'S HOMOGENEOUS FUNCTION THEOREM [5]). *A continuously differentiable function h is positive homogeneous of degree k if and only if*

$$x\frac{\partial h(x)}{\partial x} = kh(x). \tag{14}$$

B PROFIT MAXIMIZATION FOR SPS

In the following, we prove the results in Section 4.2.

We first list down all possible cases for an SP's expected payment and expected revenue when c_b and c_d fall in $[0, c_{max}]$ and

$[c_{max}, +\infty)$, respectively:

$$\mathbb{E}(Y)=\begin{cases} p_b c_b, & c_d \geq c_b > c_{max} \quad (15a)\\ p_b c_b + \int_{c_b}^{c_{max}} p_d(c-c_b)f(c)\mathrm{d}c, & c_d \geq c_{max} \geq c_b \quad (15b)\\ p_b c_b + \int_{c_b}^{c_d} p_d(c-c_b)f(c)\mathrm{d}c & \\ \quad + \int_{c_d}^{c_{max}} p_d(c_d-c_b)f(c)\mathrm{d}c, & c_{max} > c_d \geq c_b \quad (15c)\end{cases}$$

$$\mathbb{E}(R)=\begin{cases} \int_{c_{min}}^{c_{max}} R(c,1)f(c)\mathrm{d}c, & c_d \geq c_{max} \quad (16a)\\ \int_{c_{min}}^{c_d} R(c,1)f(c)\mathrm{d}c + \int_{c_d}^{c_{max}} R(c,\frac{c_d}{c})f(c)\mathrm{d}c, & c_d < c_{max} \quad (16b)\end{cases}$$

Thus, the expected profit given by $\mathbb{E}(P) = \mathbb{E}(R) - \mathbb{E}(Y)$ also has three different cases for $c_d \geq c_b > c_{max}$, $c_d \geq c_{max} \geq c_b$, and $c_{max} > c_d \geq c_b$, respectively. To prove the statements in Section 4.2, we show that the maximum expected profit for the third case is more than those for the first and second cases if SPs are GD-profitable.

When $c_d \geq c_b > c_{max}$, the expected profit $\mathbb{E}_1(P)$ is (16a) minus (15a). It is straightforward to observe that $c_d^\star = c_b^\star = c_{max}$ maximizes $\mathbb{E}(P)$.

When $c_d \geq c_{max} \geq c_b$, the expected profit $\mathbb{E}_2(P)$ is (16a) minus (15b). In this case, we still determine c_b as $c_d^\star = c_{max}$. We apply the Leibniz's rule to take partial derivative of $\mathbb{E}_2(P)$ with regard to c_b:

$$\frac{\partial \mathbb{E}_2(P)}{\partial c_b} = -p_b + \int_{c_b}^{c_{max}} p_d f(c)\mathrm{d}c. \tag{17}$$

We further take partial second-order derivative of $\mathbb{E}_2(P)$ with regard to c_b: $\partial^2\mathbb{E}_2(P)/\partial c_b^2 = -p_d f(c_d) \leq 0$, and find out that $\mathbb{E}_2(P)$ is concave of c_b. To maximize $\mathbb{E}_2(P)$, we set (17) to zero and obtain

$$\int_{c_b^\star}^{c_{max}} f(c)\mathrm{d}c = \frac{p_b}{p_d}. \tag{18}$$

Note that $c_b^\star > c_{min}$ since $p_b < p_d$ and $\int_{c_{min}}^{c_{max}} f(c)\mathrm{d}c = 1$, we can compare $\mathbb{E}_1^\star(P)$ and $\mathbb{E}_2^\star(P)$:

$$\begin{aligned}\mathbb{E}_2^\star(P) - \mathbb{E}_1^\star(P) &\overset{(a)}{=} p_b c_{max} - \int_{c_b^\star}^{c_{max}} p_d c f(c)\mathrm{d}c \\ &\overset{(b)}{\geq} p_d c_{max}\left(\frac{p_b}{p_d} - \int_{c_b^\star}^{c_{max}} f(c)\mathrm{d}c\right) = 0,\end{aligned} \tag{19}$$

where (a) is by substituting (18), and (b) is due to $c_{max} \geq c$ for $c \in [c_b^\star, c_{max}]$. Thus, $\mathbb{E}_2^\star(P) \geq \mathbb{E}_1^\star(P)$.

When $c_{max} > c_d \geq c_b$, the expected profit $\mathbb{E}_3(P)$ is calculated by (16b) minus (15c). We again apply the Leibniz's rule to take partial derivative of $\mathbb{E}_3(P)$ with respect to c_b and c_d, respectively:

$$\frac{\partial \mathbb{E}_3(P)}{\partial c_b} = -p_b + \int_{c_b}^{c_d} p_d f(c)\mathrm{d}c + \int_{c_d}^{c_{max}} p_d f(c)\mathrm{d}c, \tag{20}$$

$$\begin{aligned}\frac{\partial \mathbb{E}_3(P)}{\partial c_d} &= \int_{c_d}^{c_{max}} \frac{\partial R(c, c_d/c)}{\partial c_d} f(c)\mathrm{d}c - \int_{c_d}^{c_{max}} p_d f(c)\mathrm{d}c \\ &= \int_{c_d}^{c_{max}} \frac{k}{c_d} R\left(c, \frac{c_d}{c}\right) f(c)\mathrm{d}c - \int_{c_d}^{c_{max}} p_d f(c)\mathrm{d}c,\end{aligned} \tag{21}$$

where Theorem A.2 enables the equality in (21). Note that the expression in (20) is the same as (17) and thus $\mathbb{E}_3(P)$ is also concave of c_b. We can then obtain (9).

For $c_d \in [c_b, c_{max}]$, the conditions in (4) and (5) guarantee $\frac{\partial \mathbb{E}_3(P)}{\partial c_d}\big|_{c_d=c_b} > 0$ and $\frac{\partial \mathbb{E}_3(P)}{\partial c_d}\big|_{c_d=c_{max}} < 0$, respectively. Thus, by setting (21) to zero, there exists at least one critical point that maximizes $\mathbb{E}_3(P)$ and is just (10):

$$\int_{c_d^\star}^{c_{max}} p_d f(c)\mathrm{d}c = \int_{c_d^\star}^{c_{max}} \frac{k}{c_d^\star} R\left(c, \frac{c_d^\star}{c}\right) f(c)\mathrm{d}c. \tag{22}$$

Note that (9) for $c_{max} > c_d \geq c_b$ is the same as (18) for $c_d \geq c_{max} \geq c_b$. We next prove $\mathbb{E}_3^\star(P)$ is larger than $\mathbb{E}_2^\star(P)$ if $R(c,\theta)$ is positive homogeneous of degree $k \in (0,1)$ in terms of θ:

$$\begin{aligned}&\mathbb{E}_3^\star(P) - \mathbb{E}_2^\star(P) \\ &= \int_{c_d^\star}^{c_{max}} \left(1 - \frac{c_d^\star}{c}\right)\left(p_d c - \frac{R(c,1) - R\left(c, \frac{c_d^\star}{c}\right)}{1 - \frac{c_d^\star}{c}}\right) f(c)\mathrm{d}c \\ &\overset{(c)}{\geq} \int_{c_d^\star}^{c_{max}} (c - c_d^\star)\left(p_d - \frac{k}{c_d^\star} R\left(c, \frac{c_d^\star}{c}\right)\right) f(c)\mathrm{d}c \overset{(d)}{\geq} 0,\end{aligned} \tag{23}$$

where (c) is due to $R(c,1) = \left(\frac{c}{c_d^\star}\right)^k R\left(c, \frac{c_d^\star}{c}\right)$ and $\frac{(c/c_d^\star)^k - 1}{1 - c_d^\star/c} \leq k\frac{c}{c_d^\star}$ for $k \in (0,1)$, and (d) is due to $p_d \geq \int_{c_d^\star}^{c_{max}} \left(\frac{k}{c_d^\star}\right) R\left(c, \frac{c_d^\star}{c}\right) f(c)\mathrm{d}c$ inferred from (10).

To summarize the above discussion, with the conditions in Proposition 4.2 satisfied, we have $\mathbb{E}_3^\star(P) \geq \mathbb{E}_2^\star(P) \geq \mathbb{E}_1^\star(P)$, i.e., GD-profitable SPs' maximum expected profit can be achieved when $c_{max} > c_d > d_b > c_{min}$ with the optimal $c_b^\star$ and $c_d^\star$ that satisfy (9) and (10).

C IMPACT OF PRICE ON REQUESTED CAPACITY

In this section, we prove the results in Section 4.3.

We first prove Proposition 4.4. Since the left-hand side of (10) monotonically decreases of $c_b^\star$, it is easy to observe that $c_b^\star$ decreases with p_b and increases with p_d.

To prove the monotonic decrease of $c_d^\star$ with p_d, we rewrite (10) as

$$p_d = \left(\int_{c_d^\star}^{c_{max}} \frac{k}{c_d^\star} R(c, c_d^\star/c) f(c)\mathrm{d}c\right)\left(\int_{c_d^\star}^{c_{max}} f(c)\mathrm{d}c\right)^{-1}. \tag{24}$$

We let the right-hand side of (24) be $g(c_d^\star)$. By taking the first-order derivative of $g(c_d^\star)$ in terms of $c_d^\star$, we have

$$\begin{aligned}&\partial g(c_d^\star)/\partial c_d^\star \\ &= \left[\left(\int_{c_d^\star}^{c_{max}} k(k-1){c_d^\star}^{-2} R\left(c, \frac{c_d^\star}{c}\right) f(c)\mathrm{d}c\right)\int_{c_d^\star}^{c_{max}} f(c)\mathrm{d}c \right. \\ &\left. + \frac{k}{c_d^\star} f(c_d^\star) \int_{c_d^\star}^{c_{max}} \left(R\left(c, fracc_d^\star c\right) - R(c_d^\star, 1)\right) f(c)\mathrm{d}c\right] \\ &\times \left(\int_{c_d^\star}^{c_{max}} f(c)\mathrm{d}c\right)^{-2}.\end{aligned} \tag{25}$$

Since $k < 1$ and $\frac{\partial R(c, c_d^\star/c)}{\partial c} = -\frac{c_d^\star}{c^2}\frac{\partial R(c, c_d^\star/c)}{\partial (c_d^\star/c)} \leq 0$, we have $\frac{\partial g(c_d^\star)}{\partial c_d^\star} < 0$. Therefore, the decrease of the right-hand side of (24) implies that larger p_d leads to smaller $c_d^\star$.

Next, we prove Corollary 4.5. Suppose when p_d increases by δ_p ($\tilde{p}_d = (1+\delta_p)p_d$), the optimal total capacity $c_d^\star$ decreases by δ_d ($\tilde{c}_d^\star = (1-\delta_d)c_d^\star$). Thus, the following two equalities hold:

$$\int_{c_d^\star}^{c_{max}} \frac{k}{c_d^\star} R\left(c, \frac{c_d^\star}{c}\right) f(c)\mathrm{d}c = \int_{c_d^\star}^{c_{max}} p_d f(c)\mathrm{d}c, \tag{26}$$

$$\begin{aligned} &\int_{(1-\delta_d)c_d^\star}^{c_{max}} \frac{k}{(1-\delta_d)c_d^\star} R\left(c, \frac{(1-\delta_d)c_d^\star}{c}\right) f(c)\mathrm{d}c \\ &= \int_{(1-\delta_d)c_d^\star}^{c_{max}} (1+\delta_p)p_d f(c)\mathrm{d}c. \end{aligned} \tag{27}$$

Since $R(c,\theta)$ is positive homogeneous of degree k in terms of θ, we can derive from (27) that

$$\begin{aligned} &\frac{(1-\delta_d)^{k-1}}{(1+\delta_p)} \int_{(1-\delta_d)c_d^\star}^{c_{max}} \frac{k}{c_d^\star} R\left(c, \frac{c_d^\star}{c}\right) f(c)\mathrm{d}c \\ &= \int_{(1-\delta_d)c_d^\star}^{c_{max}} p_d f(c)\mathrm{d}c. \end{aligned} \tag{28}$$

Due to the integral inequality $\int_a^b h(x)\mathrm{d}x \geq \int_c^b h(x)\mathrm{d}x$ for $a < c$, the difference between the left-hand sides of the equalities in (28) and (26) is larger than:

$$\left(\frac{(1-\delta_d)^{k-1}}{(1+\delta_p)} - 1\right) \int_{c_d^\star}^{c_{max}} \frac{k}{c_d^\star} R\left(c, \frac{c_d^\star}{c}\right) f(c)\mathrm{d}c. \tag{29}$$

We see that the difference between the right-hand sides of the equalities in (28) and (26) is positive, in order for which to hold, (29) must also be positive, leading to $\frac{(1-\delta_d)^{k-1}}{(1+\delta_p)} \geq 1$, i.e., $\delta_d \geq 1-(1+\delta_p)^{-\frac{1}{1-k}}$, for $k \in (0,1)$.

Applying similar approach, we can also prove that: when p_d decreases by $\delta_p \in (0,1)$ ($\tilde{p}_d = (1-\delta_p)p_d$), the optimal total capacity $c_d^\star$ increases by $\delta_d \in (0,1)$ ($\tilde{c}_d^\star = (1+\delta_d)c_d^\star$), where $\delta_d \geq (1-\delta_p)^{-\frac{1}{1-k}} - 1$.

REFERENCES

[1] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. 2014. The Rise of RaaS: The Resource-as-a-service Cloud. *Commun. ACM* 57, 7 (July 2014), 76–84.

[2] Amazon Web Services. 2017. Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. (2017). Accessed: 2017-4-15.

[3] Amazon Web Services. 2017. Amazon EC2 Service Limits. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-resource-limits.html. (2017). Accessed: 2017-4-13.

[4] Amazon Web Services. 2017. Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/. (2017). Accessed: 2017-5-1.

[5] Tom M. Apostol. 1967. Calculus, 2nd. ed. *Waltham, Massachusetts: Blaisdel* (1967).

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (April 2010), 50–58.

[7] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. 2016. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 217–232.

[8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *SOSP*. ACM.

[9] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. 2013. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Vol. 8. 1–154 pages.

[10] Luiz André Barroso and Urs Hölzle. 2007. The case for energy-proportional computing. *Computer* 40, 12 (2007).

[11] Bruce R Beattie, Charles Robert Taylor, and Myles James Watts. 1985. *The economics of production*. Wiley New York.

[12] Mathias Björkqvist, Sebastiano Spicuglia, Lydia Chen, and Walter Binder. 2013. QoS-aware service VM provisioning in clouds: Experiences, models, and cost analysis. In *International Conference on Service-Oriented Computing*. Springer, 69–83.

[13] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *Commun. ACM* 59, 5 (April 2016), 50–57.

[14] Marcus Carvalho, Walfredo Cirne, Francisco Brasileiro, and John Wilkes. 2014. Long-term SLOs for Reclaimed Cloud Computing Resources. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '14)*. ACM, New York, NY, USA, Article 20, 13 pages.

[15] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. 2012. Optimization of resource provisioning cost in cloud computing. *IEEE Transactions on Services Computing* 5, 2 (2012), 164–177.

[16] Yuan Chen, Subu Iyer, Xue Liu, Dejan Milojicic, and Akhil Sahai. 2007. SLA Decomposition: Translating Service Level Objectives to System Level Thresholds. In *ICAC*. IEEE.

[17] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. 2015. Energy-Efficient Resource Allocation and Provisioning Framework for Cloud Data Centers. *IEEE Transactions on Network and Service Management* 12, 3 (Sept 2015), 377–391.

[18] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. 2013. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–32.

[19] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. 2015. IaaS reserved contract procurement optimisation with load prediction. *Future Generation Computer Systems* 53 (2015), 13 – 24.

[20] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. 2015. Utilization Prediction Aware VM Consolidation Approach for Green Cloud Computing. In *2015 IEEE 8th International Conference on Cloud Computing*. 381–388.

[21] Dror G. Feitelson. 2015. From Repeatability to Reproducibility and Corroboration. *SIGOPS Oper. Syst. Rev.* 49, 1 (Jan. 2015), 3–11.

[22] Daniel Fleder, Kartik Hosanagar, and Andreas Buja. 2010. Recommender Systems and Their Effects on Consumers: The Fragmentation Debate. In *Proceedings of the 11th ACM Conference on Electronic Commerce (EC '10)*. ACM, New York, NY, USA, 229–230.

[23] Rahul Ghosh and Vijay K. Naik. 2012. Biting Off Safely More Than You Can Chew: Predictive Analytics for Resource Over-Commit in IaaS Cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*. 25–32.

[24] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *2010 International Conference on Network and Service Management*. 9–16.

[25] Ajay Gulati, Ganesha Shanmuganathan, Anne Holler, and Irfan Ahmad. 2011. Cloud-scale Resource Management: Challenges and Techniques. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '11)*. USENIX Association, 3–3.

[26] Doug Hoyte and contributors. [n. d.]. vmtouch - the Virtual Memory Toucher. https://github.com/hoytech/vmtouch. ([n. d.]). Accessed: 2017-05-04.

[27] Zhe Huang, S. Matthew Weinberg, Liang Zheng, Mung Chiang, and Carlee-Joe Wong. 2017. Discovering Valuations and Enforcing Truthfulness in a Deadline-Aware Scheduler. In *Proceedings of IEEE INFOCOM*.

[28] Juncheng Jia, Qian Zhang, Qin Zhang, and Mingyan Liu. 2009. Revenue Generation for Truthful Spectrum Auction in Dynamic Spectrum Access. In *Proceedings of the Tenth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '09)*. ACM, New York, NY, USA, 3–12.

[29] Hai Jin, Xinhou Wang, Song Wu, Sheng Di, and Xuanhua Shi. 2015. Towards Optimized Fine-Grained Pricing of IaaS Cloud Platform. *IEEE Transactions on Cloud Computing* 3, 4 (Oct 2015), 436–448.

[30] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX, GA, 117–134.

[31] H Youn Kim. 1992. The translog production function and variable returns to scale. *The Review of Economics and Statistics* (1992), 546–552.

[32] Cristian Klein. [n. d.]. Closed and open HTTP traffic generator. https://github.com/cloud-control/httpmon. ([n. d.]). Accessed: 2017-05-04.

[33] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. 2014. Brownout: Building More Robust Cloud Applications. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 700–711.
[34] Andreas Kohne, Damian Pasternak, Lars Nagel, and Olaf Spinczyk. 2016. Evaluation of SLA-based Decision Strategies for VM Scheduling in Cloud Data Centers. In *Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms (CrossCloud '16)*. ACM, New York, NY, USA, Article 6, 5 pages.
[35] Andreas Kohne, Marc Spohr, Lars Nagel, and Olaf Spinczyk. 2014. FederatedCloudSim: A SLA-aware Federated Cloud Simulation Framework. In *Proceedings of the 2nd International Workshop on CrossCloud Systems (CCB '14)*. ACM, New York, NY, USA, Article 3, 5 pages.
[36] Ewnetu Bayuh Lakew, Cristian Klein, Francisco Hernandez-Rodriguez, and Erik Elmroth. 2015. Performance-based service differentiation in clouds. In *Proceedings of IEEE/ACM Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 505–514.
[37] Ewnetu Bayuh Lakew, Alessandro Vittorio Papadopoulos, Martina Maggio, Cristian Klein, and Erik Elmroth. 2017. KPI-agnostic Control for Fine-Grained Vertical Elasticity. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 589–598.
[38] Lawrence J Lau. 1972. Profit functions of technologies with multiple inputs and outputs. *The Review of Economics and Statistics* (1972), 281–289.
[39] Min Lee, A. S. Krishnakumar, P. Krishnan, Navjot Singh, and Shalini Yajnik. 2010. Supporting Soft Real-time Tasks in the Xen Hypervisor. In *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '10)*. ACM, New York, NY, USA, 97–108.
[40] Young Choon Lee and Albert Y. Zomaya. 2012. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing* 60, 2 (2012), 268–280.
[41] Jacob Leverich and Christos Kozyrakis. 2014. Reconciling High Server Utilization and Sub-millisecond Quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. ACM, New York, NY, USA, Article 4, 14 pages.
[42] Zheng Li, He Zhang, Liam O'Brien, Shu Jiang, You Zhou, Maria Kihl, and Rajiv Ranjan. 2016. Spot pricing in the Cloud ecosystem: A comparative investigation. *Journal of Systems and Software* 114 (2016), 1 – 19.
[43] Xiao-Fang Liu, Zhi-Hui Zhan, Ke-Jing Du, and Wei-Neng Chen. 2014. Energy Aware Virtual Machine Placement Scheduling in Cloud Computing Based on Ant Colony Optimization Approach. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, New York, NY, USA, 41–48.
[44] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 450–462.
[45] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing* 12, 4 (2014), 559–592.
[46] Nguyen Cong Luong, Ping Wang, Dusit Niyato, Yonggang Wen, and Zhu Han. 2017. Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey. *IEEE Communications Surveys and Tutorials* 19, 2 (2017), 954–1001.
[47] Michael McKeown, Yaosheng Fu, Tri M. Nguyen, Yanqi Zhou, Jonathan Balkind, Alexey Lavrov, Mohammad Shahrad, Samuel Payne, and David Wentzlaff. 2017. Piton: A Manycore Processor for Multitenant Clouds. *IEEE Micro* 37, 2 (Mar 2017), 70–80.
[48] Jeonghoon Mo and Jean Walrand. 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking* 8, 5 (2000), 556–567.
[49] Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. 2013. AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC '13)*. USENIX, San Jose, CA, 69–82.
[50] Joseph Lee Rodgers and W. Alan Nicewander. 1988. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician* 42, 1 (1988), 59–66.
[51] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. 2006. Open Versus Closed: A Cautionary Tale. In *NSDI*. USENIX.
[52] Fred C Schweppe, Michael C Caramanis, Richard D Tabors, and Roger E Bohn. 2013. *Spot pricing of electricity*. Springer Science & Business Media.
[53] Soumya Sen, Carlee Joe-Wong, Sangtae Ha, and Mung Chiang. 2013. A survey of smart data pricing: Past proposals, current plans, and future trends. *ACM Computing Surveys (CSUR)* 46, 2 (2013), 15.
[54] Mohammad Shahrad and David Wentzlaff. 2016. Availability Knob: Flexible User-Defined Availability in the Cloud. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC '16)*. ACM, New York, NY, USA, 42–56.
[55] Supreeth Shastri, Amr Rizk, and David Irwin. 2016. Transient Guarantees: Maximizing the Value of Idle Cloud Capacity. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. 992–1002.
[56] Siqi Shen, Vincent van Beek, and Alexandru Iosup. 2015. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 465–474.
[57] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SoCC '11)*. ACM, New York, NY, USA, Article 5, 14 pages.
[58] Weijie Shi, Linquan Zhang, Chuan Wu, Zongpeng Li, and Francis Lau. 2014. An online auction framework for dynamic resource provisioning in cloud computing. *Proceedings of ACM SIGMETRICS* (2014).
[59] Christopher Stewart, Terence Kelly, and Alex Zhang. 2007. Exploiting Nonstationarity for Performance Prediction. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07)*. ACM, New York, NY, USA, 31–44.
[60] Christopher Stewart and Kai Shen. 2005. Performance modeling and system management for multi-component online services. In *NSDI*. USENIX, 71–84.
[61] Thomas Stockhammer. 2011. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, New York, NY, USA, 133–144.
[62] Guibin Tian and Yong Liu. 2012. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. ACM, New York, NY, USA, 109–120.
[63] Luis Tomás, Cristian Klein, Johan Tordsson, and Francisco Hernández-Rodríguez. 2014. The Straw that Broke the Camel's Back: Safe Cloud Overbooking with Application Brownout. In *2014 International Conference on Cloud and Autonomic Computing*. 151–160.
[64] Luis Tomás and Johan Tordsson. 2013. Improving Cloud Infrastructure Utilization Through Overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC '13)*. ACM, New York, NY, USA, Article 5, 10 pages.
[65] Nedeljko Vasić, Dejan Novaković, Svetozar Miučin, Dejan Kostić, and Ricardo Bianchini. 2012. DejaVu: Accelerating Resource Allocation in Virtualized Environments. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII)*. ACM, New York, NY, USA, 423–436.
[66] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*. ACM, New York, NY, USA, Article 18, 17 pages.
[67] Jan Vitek. 2014. The Case for the Three R's of Systems Research: Repeatability, Reproducibility and Rigor. In *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '14)*. ACM, New York, NY, USA, 115–116.
[68] Cheng Wang, Bhuvan Urgaonkar, Aayush Gupta, Lydia Y. Chen, Robert Birke, and George Kesidis. 2016. Effective Capacity Modulation as an Explicit Control Knob for Public Cloud Profitability. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. 95–104.
[69] Qian Wang, Kui Ren, and Xiaoqiao Meng. 2012. When cloud meets eBay: Towards effective pricing for cloud computing. In *2012 Proceedings IEEE INFOCOM*. 936–944.
[70] Hong Xu and Baochun Li. 2013. Dynamic Cloud Pricing for Revenue Maximization. *IEEE Transactions on Cloud Computing* 1, 2 (July 2013), 158–171.
[71] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 607–618.
[72] Xiaomeng Yi, Fangming Liu, Zongpeng Li, and Hai Jin. 2016. Flexible Instance: Meeting Deadlines of Delay Tolerant Jobs in The Cloud with Dynamic Pricing. In *Proceedings of IEEE ICDCS*.
[73] Min Sang Yoon, Ahmed E. Kamal, and Zhengyuan Zhu. 2016. Requests Prediction in Cloud with a Cyclic Window Learning Algorithm. In *2016 IEEE Globecom Workshops (GC Wkshps)*. 1–6.
[74] Jian Zhao, Hongxing Li, Chuan Wu, Zongpeng Li, Zhizhong Zhang, and Francis C. M. Lau. 2014. Dynamic pricing and profit maximization for the cloud with geo-distributed data centers. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 118–126.
[75] Liang Zheng, Carlee Joe-Wong, Jiasi Chen, Christopher G. Brinton, Chee Wei Tan, and Mung Chiang. 2017. Economic Viability of a Virtual ISP. In *Proceedings of IEEE INFOCOM*.
[76] Liang Zheng, Carlee Joe-Wong, Chee Wei Tan, Mung Chiang, and Xinyu Wang. 2015. How to bid the cloud. *Proceedings of ACM SIGCOMM* (2015).
[77] Wei Zheng, Ricardo Bianchini, G. John Janakiraman, Jose Renato Santos, and Yoshio Turner. 2009. JustRunIt: experiment-based management of virtualized data centers. In *USENIX Annual Technical Conference*. 18–28.
[78] Timothy Zhu, Daniel S. Berger, and Mor Harchol-Balter. 2016. SNC-Meister: Admitting More Tenants with Tail Latency SLOs. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC '16)*. ACM, New York, NY, USA, 374–387.