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  
wentzlaff@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

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.

3 SYSTEM OVERVIEW

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.
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 $c = a \lambda$, where $\lambda$ is a measure of load — such as number of users — and $a$ 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 reasons1. 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.

---

1Amazon EC2, for instance, sets default limits on its resources on a per-region basis [3]
We described our model which consists of reserved (c) 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:

\[ p_b < p_d. \] (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 as well as total capacity 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_d and c_{max} 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

\[ \theta = \frac{c_d}{c}, \quad (c_d < c), \] (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 \( \theta \) 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 \( \theta \) 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 \( \theta \):
\[ R(c, \theta_1) \geq R(c, \theta_2) \text{ if } \theta_1 \geq \theta_2 \]
\[ R(c_1, \theta) \geq R(c_2, \theta) \text{ if } c_1 \geq c_2. \]

(ii) Positive homogeneity\(^2\) of degree \( k \) in terms of \( \theta \):
\[ R(c, \lambda \theta) = \lambda^k R(c, \theta). \]

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 \( \theta \) increases by proportion \( \lambda \), revenue increases by proportion \( \lambda^k \).

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

\[ R(c, \theta) = yc^\beta \] (3)

where \( y \) 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 \( \theta \) and \( c \). When \( y = 1/(1-\alpha) \) and \( k = \beta = 1-\alpha \) for \( \alpha \in [0, 1) \), (3) is the commonly-used \( \alpha \)-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 \( c_d < c_{max} \). When \( c > c_d \), 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 \( \theta \). 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 \( c_d \) 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.
We suppose 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, \( c^*_b \) and \( c^*_d \).

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

\[
\int_{c_{d}}^{c_{\text{max}}} \frac{k}{c_{b}} R\left(\frac{c_{b}}{c_{d}} f(c) dc \right) > p_b, \quad (4)
\]

\[
\int_{c_{d}}^{c_{\text{max}}} \frac{k}{c_{d}} R(c_{\text{max}} - 1) < p_d, \quad (5)
\]

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

As a remark, we observe: the condition in (4) implies that reserved price is less than the expected unit revenue above \( c_d \), 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 \( c_{\text{max}} < c_d \). 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 \( c_{\text{max}} > c_d > c_b > c_{\text{min}} \), the expected payment for an SP can be calculated by

\[
\mathbb{E}(Y) = p_b c_b + \int_{c_b}^{c_d} p_d(c - c_b)f(c) dc + \int_{c_d}^{c_{\text{max}}} p_d(c_d - c_b)f(c) dc, \quad (6)
\]

and its expected revenue can be calculated by

\[
\mathbb{E}(R) = \int_{c_{\text{min}}}^{c_d} R(c, 1) f(c) dc + \int_{c_d}^{c_{\text{max}}} R(c, c_d/c)f(c) dc, \quad (7)
\]

leading to the expected profit of

\[
\mathbb{E}(P) = \mathbb{E}(R) - \mathbb{E}(Y). \quad (8)
\]

where \( c > c_d \): 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.

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

\[
\int_{c_b}^{c_{\text{max}}} \frac{k}{c_b} R\left(\frac{c_b}{c_d} f(c) dc \right) > p_b, \quad (4)
\]

\[
\int_{c_d}^{c_{\text{max}}} \frac{k}{c_d} R(c_{\text{max}} - 1) < p_d, \quad (5)
\]

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

As a remark, we observe: the condition in (4) implies that reserved price is less than the expected unit revenue above \( c_d \), 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 \( c_{\text{max}} < c_d \). 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 \( c_{\text{max}} > c_d > c_b > c_{\text{min}} \), the expected payment for an SP can be calculated by

\[
\mathbb{E}(Y) = p_b c_b + \int_{c_b}^{c_d} p_d(c - c_b)f(c) dc + \int_{c_d}^{c_{\text{max}}} p_d(c_d - c_b)f(c) dc, \quad (6)
\]

and its expected revenue can be calculated by

\[
\mathbb{E}(R) = \int_{c_{\text{min}}}^{c_d} R(c, 1) f(c) dc + \int_{c_d}^{c_{\text{max}}} R(c, c_d/c)f(c) dc, \quad (7)
\]

leading to the expected profit of

\[
\mathbb{E}(P) = \mathbb{E}(R) - \mathbb{E}(Y). \quad (8)
\]


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 ($c$) to the requested total capacity ($c^d$):

$$u_c(t) = \frac{c(t)}{c^d} = \frac{\min(c(t), c^d)}{c^d}.$$  

(11)

Here, $c(t)$ is the provisioned capacity which is bounded by $c^d$, whereas $c$ 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 ($c^*_b$) and total capacity ($c^*_p$) 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 $c^*_b$ and $c^*_p$ in (9) and (10) with regard to $p_b$ and $p_d$:

**Proposition 4.4.** The optimal reserved capacity $c^*_b$, as given in (9), has a direct relationship with $\frac{\bar{c}d}{p^*_b}$, while the optimal total capacity $c^*_p$, as given in (10), has an inverse relationship with $p_d$.

As we can observe from (11), the key to the IP’s utilization rate lies in the value of $c^*_p$. Leveraging Proposition 4.4, we further quantify this monotonic dependency between $c^*_p$ 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., $\delta p = (1 + \delta p)p_d$, then the optimal total capacity $c^*_p$ decreases/increases by $\delta d \in (0, 1)$, i.e., $c^*_p = (1 - \delta q)c^*_d$, where $\delta d \geq 1 - (1 + \delta p)^{1/k}$.

The above proposition and corollary, proof of which can be found in Appendix C, address the monotonic dependency 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 $1 - (1 + \delta p)^{1/k}$ and $(1 - \delta p)^{1/k} - 1$ of the changes in the total capacity $c^*_p$ due to both positive and negative changes in $p_d$ are increasing functions of $k \in (0, 1)$. Thus, Corollary 4.5 can be further relaxed: an increasing/decreasing change of the price $p_d$ by $\delta p$ must lead to a decreasing/increasing change of $\delta d$ in $c^*_p$ that satisfies $\delta d \geq \delta p/(1 + \delta p)$. 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. However, 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.

<table>
<thead>
<tr>
<th>Desired objective</th>
<th>Required capacity</th>
<th>Global price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase utilization</td>
<td>$\left(c^<em>_b \uparrow, c^</em>_d \uparrow\right)$</td>
<td>$(\frac{\bar{c}d}{p^*_b} \uparrow, p_d \downarrow)$</td>
</tr>
<tr>
<td>Increase effective utilization</td>
<td>$\left(c^<em>_b \uparrow, c^</em>_d \downarrow\right)$</td>
<td>$(\frac{\bar{c}d}{p^*_b} \uparrow, p_d \downarrow)$</td>
</tr>
</tbody>
</table>

**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.](image)

**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-nocompliant 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
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

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

where $c$ is the total CPU capacity demand in VM unit (assumed 2926 MHz for a VM) and $\theta$ 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 optimal reserved capacity, $c_b^*$, using (9), the GD-compliant SP can also select optimal total capacity ($c_f^*$) values less than $c_{max}$. In contrast, the GD-noncompliant SP needs to avoid capacity shortage ($c_f^*$ 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 $c_b^*$ 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 \( \frac{\text{Revenue}}{\text{Cost}} \) ratio and sensitivity to degradation. While the \( \frac{\text{Revenue}}{\text{Cost}} \) 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.

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 \( \frac{L}{p_d} \) due...
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 (\( c^*_p \)) and the ratio of \( \frac{y}{p_d} \) influences the optimal total capacity (\( c^*_t \)). This means that if \( p_b, p_d \), and \( y \) (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 \( \frac{y}{p_d} \) 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 100% 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 httpron [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 $2 \times 10^{-5}$, so that the load fits within the 30 cores of our testbed. After compression, 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 $p_h$ 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 $c_d \geq c_h = 1.0$.

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 $c_h = c_d = 1.0$. 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 $c_h$ 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, $c_h$ 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 $c_h$ 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.

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

---

4https://github.com/cristiklein/gdinc-experiment
Incentivizing Self-Capping to Increase Cloud Utilization

When to enable the optional code. With proper coordination be-

Although Chaisiri et al. [15] suggested leveraging on-demand and

Amazon EC2’s spot instances [4, 42]. As we believe current solutions (e.g. Cyclic Window Learning Al-

nor 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 im-

prove 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 claimed 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 dynam- ics [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 opera- tional perspective, but their primary objectives are either job sched- uling [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 de-

mand function. We did not introduce new prediction mechanisms as we believe current solutions (e.g. Cyclic Window Learning Al-

gorithm [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 utili-

zation 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 sup-

ported 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 $h: \mathbb{R}^+ \rightarrow \mathbb{R}$ is positive homogeneous of degree $k$ if

$$h(\lambda x) = \lambda^k h(x).$$

We point out a useful property of positive homogeneous func-

tions. 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

$$x \frac{dh(x)}{dx} = kh(x).$$

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 pay-

ment and expected revenue when $c_b$ and $c_d$ fall in $[0, c_{max}]$ and
maximizes the expression in (20) is the same as (17) and thus the Leibniz’s rule to take partial derivative of \( \mathbb{E}(Y) \) is (16a) minus (15a). It is straightforward to observe that \( c_{\text{d}}^* = c_t^* = c_{\text{max}} \) maximizes \( \mathbb{E}(P) \).

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

\[
\frac{\partial \mathbb{E}_2(P)}{\partial c_{\text{d}}^*} = -p_d + \int_{c_{\text{d}}^*}^{c_{\text{max}}} p_d f(c) \, dc. \tag{18}
\]

We further take partial second-order derivative of \( \mathbb{E}_2(P) \) with regard to \( c_{\text{d}}^* \):

\[
\frac{\partial^2 \mathbb{E}_2(P)}{\partial c_{\text{d}}^*} = -p_d f(c_{\text{d}}^*) \leq 0, \quad \text{and find out that } \mathbb{E}_2(P) \text{ is concave of } c_{\text{d}}. \quad \text{To maximize } \mathbb{E}_2(P), \text{ we set (17) to zero and obtain}
\]

\[
\int_{c_{\text{d}}^*}^{c_{\text{max}}} f(c) \, dc = \frac{p_b}{p_d}. \tag{19}
\]

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

\[
\mathbb{E}_2^*(P) - \mathbb{E}_1^*(P) = \frac{\partial \mathbb{E}_2(P)}{\partial c_{\text{d}}^*} + \int_{c_{\text{d}}^*}^{c_{\text{max}}} p_d f(c) \, dc \tag{20}
\]

where (a) is by substituting, (b) is due to \( c_{\text{max}} \geq c \) for \( c \in [c_{\text{d}}^*, c_{\text{max}}] \). Thus, \( \mathbb{E}_2^*(P) \geq \mathbb{E}_1^*(P) \).

When \( c_{\text{max}} > c_{\text{d}} \), 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 \( c_{\text{d}} \), respectively:

\[
\frac{\partial \mathbb{E}_3(P)}{\partial c_b} = -p_b + \int_{c_b}^{c_{\text{max}}} p_d f(c) \, dc + \int_{c_{\text{d}}}^{c_{\text{max}}} p_d f(c) \, dc, \quad \text{respectively:}
\]

\[
\frac{\partial \mathbb{E}_3(P)}{\partial c_{\text{d}}} = \int_{c_{\text{d}}}^{c_{\text{max}}} \frac{\partial}{\partial c_{\text{d}}} R(c, t/c) f(c) \, dc - \int_{c_{\text{d}}}^{c_{\text{max}}} p_d f(c) \, dc, \tag{21}
\]

where \( \text{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) \text{ is also concave of } c_b. \text{ We can then obtain (9).}

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

\[
\int_{c_{\text{d}}^*}^{c_{\text{max}}} p_d f(c) \, dc = \int_{c_{\text{d}}^*}^{c_{\text{max}}} k \frac{R(c, c^*_d/c)}{c^*_d} f(c) \, dc. \tag{22}
\]

Note that (9) for \( c_{\text{max}} > c_d \geq c_b \) is the same as (18) for \( c_d \geq c_{\text{max}} \geq c_b \). We next prove \( \mathbb{E}_2^*(P) \) is larger than \( \mathbb{E}_1^*(P) \) if \( R(c, 0) \) is positive homogeneous of degree \( k \in (0, 1) \) in terms of \( \theta \):

\[
\mathbb{E}_2^*(P) = \mathbb{E}_1^*(P) + \int_{c_{\text{d}}^*}^{c_{\text{max}}} f(c) \, dc \left( 1 - \frac{c_{\text{d}}^*}{c} \right) \left( 1 - \frac{R(c, c^*_d/c)}{R(c, c^*_d/c)} \right) \, dc \quad \text{for } k \in (0, 1), \quad \text{and (d) is due to } p_d \geq \int_{c_{\text{d}}^*}^{c_{\text{max}}} \left( k \frac{R(c, c^*_d/c)}{c} \right) f(c) \, dc \text{ inferred from (10).}
\]

To summarize the above discussion, with the conditions in Proposition 4.2 satisfied, we have \( \mathbb{E}_2^*(P) \geq \mathbb{E}_1^*(P) \), i.e., GD-profitable SPs’ maximum expected profit can be achieved when \( c_{\text{max}} > c_d > c_{\text{min}} \) with the optimal \( c_b^* \) and \( c_d^* \) 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 \( c_b^* \), it is easy to observe that \( c_b^* \) decreases with \( p_b \) and increases with \( p_d \).

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

\[
\mathbb{P}_d = \left( \int_{c_{\text{d}}^*}^{c_{\text{max}}} \frac{k}{c_d} R(c, c_b^*/c) f(c) \, dc \right)^{-1}. \tag{23}
\]

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

\[
\frac{\partial g(c_b^*)}{\partial c_d^*} = \left( \frac{k}{c_d} \left( \int_{c_{\text{d}}^*}^{c_{\text{max}}} \frac{k}{c_d} (c_b^*/c) f(c) \, dc \right)^{-1} \right) \frac{\partial R(c, c_b^*/c)}{\partial c_d^*} \left( \int_{c_{\text{d}}^*}^{c_{\text{max}}} \frac{k}{c_d} R(c, c_b^*/c) f(c) \, dc \right)^{-1}
\]

\[
\times \left( \int_{c_{\text{d}}^*}^{c_{\text{max}}} f(c) \, dc \right)^{-2}. \tag{25}
\]

Since \( k < 1 \) and \( \frac{\partial R(c, c_b^*/c)}{\partial c_d^*} \leq 0 \), we have \( \frac{\partial g(c_b^*)}{\partial c_d^*} < 0 \). Therefore, the decrease of the right-hand side of (24) implies that larger \( p_d \) leads to smaller \( c_d^* \).
Next, we prove Corollary 4.5. Suppose when $p_d$ decreases by $\delta p$ ($\tilde{p}_d = (1 + \delta p)p_d$), the optimal total capacity $c_d^*$ decreases by $\delta_d$. Thus, the following two equalities hold:

\[
\int_{c_d^*}^{c_d^{\text{max}}} \frac{k}{c_d} R(x) \, dc = \int_{c_d^*}^{c_d^{\text{max}}} p_d f(c) dc,
\]

(26)

\[
\int_{(1-\delta_d)c_d^*}^{c_d^{\text{max}}} \frac{k}{c_d} R(x) \, dc = \int_{(1-\delta_d)c_d^*}^{c_d^{\text{max}}} (1 + \delta_p)p_d f(c) dc.
\]

(27)

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

\[
(1 - \delta_d)^{k-1} \int_{(1+\delta_p)p_d}^{c_d^{\text{max}}} \frac{k}{c_d} R(x) \, dc = \int_{(1-\delta_d)c_d^*}^{c_d^{\text{max}}} (1 + \delta_p)p_d f(c) dc.
\]

(28)

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

\[
\left(1 - \delta_d\right)^{k-1} \int_{1+\delta_p}^{c_d^{\text{max}}} \frac{k}{c_d} R(x) \, dc = f(c_d) \geq \int_{1/2}^{c_d^{\text{max}}} \frac{k}{c_d} R(x) \, dc.
\]

Thus, 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 $(1-\delta_d)^{k-1} \geq 1$, 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 $\delta_p = (0, 1)$ ($\tilde{p}_d = (1 + \delta_p)p_d$), the optimal total capacity $c_d^*$ increases by $\delta_d = (0, 1)$ ($\tilde{c}_d^* = (1 + \delta_p)c_d^*$), where $\delta_d \geq 1 - (1 - \delta_p^{\frac{1}{1-k}}) - 1$.

REFERENCES


