# The straw that broke the camel's back: safe cloud overbooking with application brownout

Luis Tomás, Cristian Klein, Johan Tordsson, Francisco Hernández-Rodríguez
Department of Computing Science, Umeå University, Sweden
Email: {luis,cristian.klein,tordsson,hernandf}@cs.umu.se

***Abstract*—Resource overbooking is an admission control technique to increase utilization in cloud environments. However, due to uncertainty about future application workloads, overbooking may result in overload situations and deteriorated performance. We mitigate this using *brownout*, a feedback approach to application performance steering, that ensures graceful degradation during load spikes and thus avoids overload. Additionally, brownout management information is included into the overbooking system, enabling the development of improved reactive methods to overload situations. Our combined brownout-overbooking approach is evaluated based on real-life interactive workloads and non-interactive batch applications. The results show that our approach achieves an improvement of resource utilization of 11 to 37 percentage points, while keeping response times lower than the set target of 1 second, with negligible application degradation.**

## I. Introduction

Combining statistical multiplexing of resource demands, server consolidation and economy of scales, cloud providers are able to offer users resources at competitive prices. Users often exaggerate the sizes of the Virtual Machines (VMs) they lease, either because the provider forces them to use pre-defined sizes [1], common practice [2], or to compensate for uncertainty [3]. Hence, a provider could practice *overbooking*: An autonomic [4] admission controller selects whether to accept a new user application or not, based on *predicted* resource utilization, which is likely smaller than the *requested* amount of resources [5]. Overbooking is beneficial both to the provider, who can gain a competitive advantage and increase profits [6], and the user, who may observe lower prices.

Unfortunately, as overbooking relies on predictions, even the most sophisticated prediction algorithms may make mistakes (see Section II-A). For example, a conservative provider may reject user requests to avoid infrastructure overload, and observe a posteriori that the resources are underutilized. This leads to revenue loss, which, on the long term, forces the provider to increase prices. Conversely, a risk-willing provider may accept a request, expecting no overload, and observe a posteriori that the infrastructure was shortly overloaded, impacting user performance.

In addition to overbooking actions, unexpected events may impact the performance: unexpected peaks — also called flash crowds — may increase the workload by up to 5 times [7]. Similarly, unexpected hardware failures in data centers are the norm rather than an exception [8], [9]. Given the large magnitude and the relatively short duration of such unexpected events, it is often economically unfeasible to provision enough capacity for such events through well known techniques such as elasticity [10] for load peaks, replication against failures, or dynamic load balancing [11], [12]. As a result, an application can *saturate*, i.e., it can become unable to serve users in a timely manner. Depending on the intended purpose of the application, user tolerable waiting time may be as high as 2 seconds [13] or as low as 100 milliseconds [14]. Hence, for users to safely benefit from overbooking, cloud providers need some kind of mechanism to withstand short, temporary overloads (due to either overbooking decisions or unexpected events). One such solution is making interactive applications more robust to these unexpected events. A promising direction is *brownout* [15], a software engineering paradigm that has been shown to make cloud application more robust to capacity shortages, such as during flash-crowds or hardware failures. Brownout self-optimizes application resource requirement through user experience degradation (see Section II-B).

Although combining overbooking and brownout may seem straight-forward, the two approaches should not be used without thorough evaluation. Indeed, the two autonomic feedback loops, belonging to the brownout application and the overbooking provider, may take conflicting decisions, which may degrade performance. By contrast, if both approaches are effectively combined, the overbooking system may take advantage of the application performance knowledge from brownout, and use both reactive and proactive methods to avoid overload situations.

In this paper, we explore two methods to combine overbooking and brownout: a naïve one and a smart one. At the core of both approaches lies a provider-operated overbooking controller that admits or rejects new applications, and a user-operated brownout controller that adapts the capacity requirements of already accepted applications. In the naïve approach, the overbooking controller bases its decision on a statically configured target resource utilization chosen by the provider, whereas in the smart approach, this parameters is self-optimized using brownout-specific monitoring information sent by the application. Our contribution is twofold:

1) We present two approaches to increase resource utilization through overbooking, while at the same time avoiding temporary overloads through brownout (Section III).
2) We evaluate the benefits of our approaches, using interactive applications, as well as non-interactive batch applications that have either steady or bursty resource requirements (Section IV).

The experimental results show that resource utilization and capacity allocated can be increased by 11 to 37 percentage points, while maintaining the response times below the upper bound, in our case 1 second, with negligible impact on user experience.

## II. Background and Related Work

In this section, we present the necessary background to understand our contribution. First we describe overbooking for cloud infrastructures, then we present the brownout paradigm for cloud application development.

### A. *Overbooking: Increasing Infrastructure Utilization*

Cloud computing is defined by on-demand access to computing resources from a shared pool. Capacity is allocated to users by leasing VMs of a certain size, determined by a number of CPU cores and amount of memory. Note that, the capacity actually used by the user may be arbitrarily smaller than the requested capacity, i.e., the size of the VM. Indeed, as several workload studies show, cloud data centers are currently poorly utilized [16]. There are several factors contributing to low data center utilization, such as cloud providers only offering predefined VM sizes, applications requiring variable amounts of resources, or users overestimating their application requirements.

One way of addressing those problems and increasing resource utilization is resource **overbooking**. In essence, the provider *allocates more capacity* than the *real capacity* of the data center [17]. In other words, a new VM is admitted although the sum of requested cores or memory exceeds the number of cores or total memory in the data center. However, such an approach may lead to resource overload and performance degradation. Therefore, besides carefully choosing how to place VMs on physical machines [18], a new resource management challenge appears: estimating the appropriate level of overbooking that can be achieved without impacting the performance of the cloud services. Admission control techniques are therefore needed to handle this trade-off between increasing resource utilization and risking performance degradation.

Examples of studies centered on evaluating the risk of resource overbooking are [17], [19] and [20]. The former performs a statistical analysis of the aggregate resource usage behavior of a group of workloads and then applies a threshold-based overbooking schema. The later two are based on Service Level Agreement (SLA) management to handle the trade-off between overbooking and risk of performance degradation, which implies payment of penalties. They calculate the probability of successfully deploying additional VMs. However, all of the cited works only consider CPU overbooking.

We presented in [21] an overbooking framework based on a long term risk evaluation of the overbooking decisions in order to avoid resource shortage in three different dimensions: CPU, memory, and I/O. This framework implements a fuzzy logic risk assessment that estimates the risk associated to each overbooking action. Fuzzy logic was chosen to avoid having to handle large amount of critical information for the risk evaluation, combined with the need to deal with uncertainty, e.g., unknown application workload in the future. The overbooking system pursues a pre-defined target utilization by accepting or rejecting new applications based on their associated fuzzy risk values. A Proportional-Integral-Derivative (PID) controller [22] is used to dynamically change a threshold for the acceptable level of risk, depending on how much the data center utilization deviates from the desired utilization level.

The presented framework tries to avoid overload situations, but these are still possible due to uncertainty and inaccurate predictions. Additionally, the impact of potential overload is hard to assess. The impact is both application- and capacity-coupled — each application may tolerate a different level of overbooking — highlighting the need of mitigation and recovery methods. One such method was proposed by Beloglazov et al. [23], where a Markov chain model and a control algorithm are used to detect overload problems in the physical servers. When a problematic situation is detected, the system migrates some VMs to less loaded resources. Another example is the Sandpiper engine presented in [24], which detects hotspots and performs the needed migration actions in order to reduce the performance degradation. Both approaches use live migration [25], which imply a performance degradation while VMs are moved. Furthermore, both resource usage and applications downtime due to migration is very hard to predict [26]. At any rate, these techniques only deal with overload localized on a few physical machines and do not provide solutions if the whole data center is overloaded. The Sandpiper framework also compares an applications-agnostic approach with another exploiting application-level statistics, demonstrating the improvements achieved by having more information about the applications. However, our previous overbooking framework lacks information about what kind of cloud applications are being provisioned and about whether or not the application meets its target performance. Therefore, we need a method that (1) allows the overbooking framework to have information about current performance of running cloud applications so that it can react to overload situations and do better overbooking, and (2) provides quick temporal solutions while the overbooking system is reconfiguring itself to adapt to the current load situation.

### B. *Brownout: Making Cloud Applications More Robust*

To allow applications to more robustly handle unexpected events and avoid saturation (i.e., high response times) due to lack of computing capacity, applications could be extended to gracefully degrade user experience and reduce resource requirements. Several solutions have been proposed [27], [28], [29], [30], but these are not applicable to cloud applications for two reasons: they are very specific to one type of application and, second, they use CPU usage to detect saturation, which is not a reliable measure for spare capacity in virtualized environments, due to hypervisor preemption of VMs, also called steal time [31].

For making cloud applications more robust, in a generic and non-intrusive manner, we proposed a new programming paradigm called **brownout** [15]. The term is inspired from brownout in electrical grids, which are intentional voltage drops used to prevent more severe blackouts. Brownout can be added to applications in three steps. First, the developer identifies which part of the request-response application can be considered optional. For example, product recommendations in e-commerce application may be discarded without impacting usability while greatly reducing the capacity requirements of the application. Second, a control knob is exported to control how often these optional computations are executed.

This knob, the **dimmer**, represents the probability of serving a query with optional content. Last, a separate component, the **brownout controller**, is added to adjust the dimmer as required to avoid saturation. More specifically, an adaptive PID controller is used to maintain maximum response times around a configured set-point, for example, 1 second. Thus, if a brownout application is close to saturation due to insufficient capacity, it reduces the dimmer, which in turn reduces the number of requests served with optional content and capacity requirements.

However, as a side-effect of controlling maximum response-time, brownout applications tend to keep resources slightly underutilized in expectation of sudden query bursts. Hence, an infrastructure hosting such an application may interpret this underutilization as a sign that the application is performing well, when in fact, the application has to operate at reduced dimmer value and only occasionally serve optional content. Therefore this fact should be considered by the cloud provider when taking its management decisions, including overbooking.

## III. BOB: Brownout OverBooking

Based on the advantages and complementary features of overbooking and brownout approaches, we propose two architectures that combine them in order to increase utilization while at the same time maintaining applications performance: a naïve one and a smart one.

### A. Naïve BOB

In the naïve version of our architecture, the existing brownout and overbooking components are simply put together with no integration effort. This allows us to evaluate the benefits of brownout inside a non-supporting data center, paving the path to an incremental deployment. Effectively, the system features two non-coordinated autonomic feedback loops (Fig. 1). At the data center level, the overbooking controller monitors resource utilization, computes an acceptable risk based on a target utilization and accepts or rejects new VMs based on the calculated acceptable risk level. On the applications level, each brownout application $i$ features a controller that monitors response times $t_i$ and adjust the dimmer $\theta_i$ accordingly, based on a target response time – note that not all the applications running inside the data center must be brownout applications.

Noteworthy is the lack of an explicit communication mechanism to coordinate the two feedback loops. The overbooking controller is unaware that the hosted applications may change their resource requirements over time and may thus assume that low VM utilization is due to over-provisioning, whereas, in fact, the application reduced its dimmer to avoid saturation due to insufficient capacity. Despite this, a careful choice of the target utilization would allow all hosted applications to execute without triggering brownout.

Besides the inconvenience of manually choosing this parameter, it is still possible that the obtained results are suboptimal. Some applications might work well with higher target utilizations, whereas others might be more sensitive, requiring the infrastructure to maintain lower target utilization.

**Algorithm 1** Target Utilization Controller

```
Configuration parameters: $duration$, update interval of output
    $TU_{min}$ and $TU_{max}$, minimum and maximum acceptable target utilization
    $incrementStep$, how much to increase utilization
 1: $TU \leftarrow 0.80$ { default target utilization }
 2: while true do
 3:     $n_i \leftarrow$ number of negative matching values sent by application
        $i$ since the last update
 4:     $num \leftarrow$ number of $n_i > 0$, i.e., number of impacted applications
 5:     if $num > 0$ then
 6:         $totalImpact = \sum_i n_i$
 7:         $decrementStep = \sqrt{\frac{totalImpact}{num*duration}}$
 8:         $TU \leftarrow TU - (TU_{max} - TU_{min}) * decrementStep$
 9:     else
10:         $TU \leftarrow TU + (TU_{max} - TU_{min}) * incrementStep$
11:     saturate $TU$ between $TU_{min}$ and $TU_{max}$
12:     send $TU$ to overbooking controller
13:     sleep for $duration$
```

### B. Smart BOB

Our aim is thus to design an architecture that self-optimizes target utilization, to maximize actual utilization while minimally triggering brownout. Building on naïve BOB, we extended it to produce an autonomic coordinated version (Fig. 1). In essence, we added an explicit communication mechanism between the brownout controllers and the overbooking controller to achieve coordination of the two feedback loops. The mechanism consists of an application level matching value and a data center level target utilization controller.

In addition to the dimmer value, the brownout controller also computes a **matching value** $m_i$, that expresses how well the application is performing with its current resource allocation. A formula for the matching value was proposed in [32], which showed good behavior when dealing with capacity auto-scaling (vertical elasticity) among competing brownout applications. Hence, we chose to maintain the same formulation for the matching value in the present admission control problem:

$$m_i = 1 - t_i / \bar{t}_i \tag{1}$$

where $t_i$ is the maximum response time over the last control interval and $\bar{t}_i$ is the target response time. The matching value abstracts application performance indicators and self-optimization, such as target response time and control strategy, from the infrastructure. The matching value hides from the infrastructure applications performance indicators, such as response-time, and how they are used in self-optimization.

At the data center level, these matching values are collected by the **Target Utilization Controller (TUC)**. Based on matching values received from all applications, the TUC periodically updates the target utilization ($TU$) that is then sent to the overbooking controller. As described in Algorithm 1, if at least one application sends a negative matching value, then the target utilization is decreased as a function of the number of negative matching values per application received since the last update. $TU$ can decrease up to a configured minimum acceptable target utilization $TU_{min}$. Otherwise, if no application sent a negative matching value, the target utilization is progressively increased until it reaches a configured maximum $TU_{max}$.

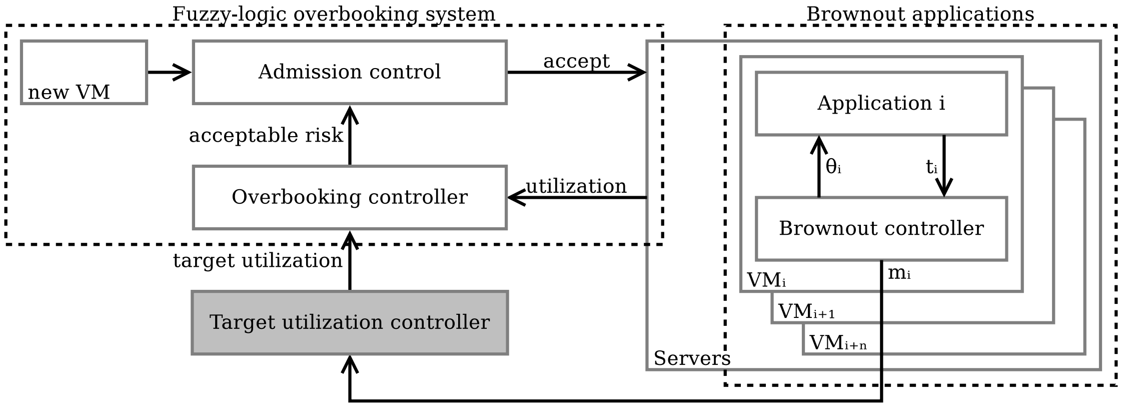


Fig. 1: Brownout OverBooking architectures. The gray component is used only in the smart architecture.

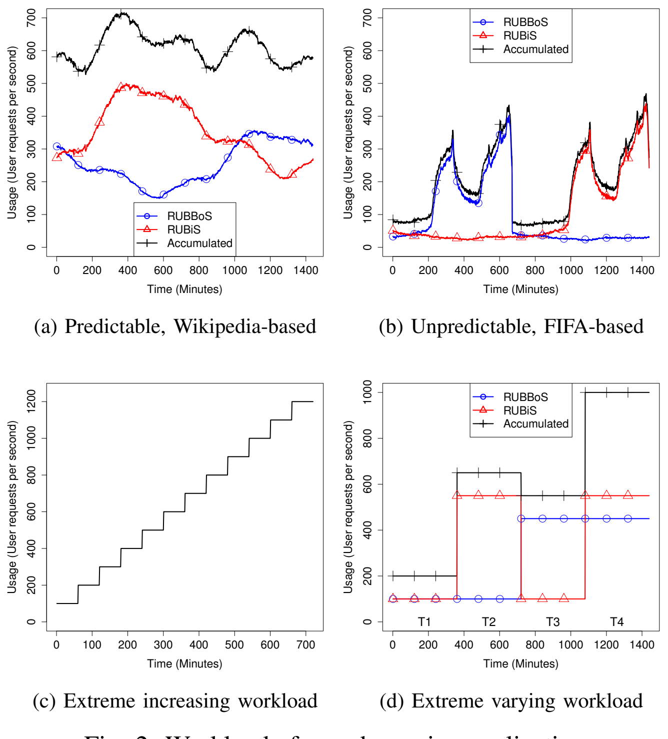
All in all, Smart BOB, besides minimizing dimmer reduction, includes applications performance at the data center decision level, enabling improved reactive methods for overload situations, that further reduce their impact. By self-optimizing the target utilization depending on applications performance, smart BOB relieves the provider from manually selecting and readjust this value. Hence, the infrastructure could adapt the degree of overbooking to the performance profiles of the hosted applications, increasing resource utilization while at the same time minimizing the amount of application brownouts.

## IV. Evaluation

The performance of our two suggested architectures is evaluated with two different brownout applications and the resource utilization is studied under four workloads – two real traces and two generated ones – which are compared in four different scenarios: no overbooking, overbooking without brownout, naïve BOB, and smart BOB.

We study web servers and investigate four metrics. The average and 95-percentile **response time** represents the time elapsed since an end-user sends the first byte of the HTTP query until the last byte of the reply is received. The **dimmer** represents the percentage of queries served with optional content and quantifies the quality of experience of the end-user interacting with a brownout application and the revenue that the cloud user may generate. **Capacity allocated** measures the sum of applications' resource requests and quantifies the revenue that the provider could obtain. Finally, the **real utilization** measures the capacity actually used by hosted applications, showing insight into missed opportunities to accept more applications. Although all the capacity dimensions were considered in the performance evaluation, the capacity allocated and real utilization metrics in the subsequent figures only refer to CPU usage. The other dimensions, memory and I/O, were prevented from overloading thanks to the fuzzy-logic overbooking controller alone, as explained in [21].

The tests were conducted on two machines, one hosting applications and one generating the workload, connected through a 100 MB link. The first machine is a server consisting of a total of 32 CPU cores (AMD Opteron$^{\text{TM}}$ 6272 at 2.1 GHz) and 56 GB of memory. KVM was used as a hypervisor and each application was deployed inside a VM. The second machine is a Ubuntu 12.04 desktop featuring a 4-core Intel Core$^{\text{TM}}$ i5 processor at 3.4 GHz and 16 GB of memory. The client queries were generated using the `httpmon` tool [33].



(a) Predictable, Wikipedia-based

(b) Unpredictable, FIFA-based

(c) Extreme increasing workload

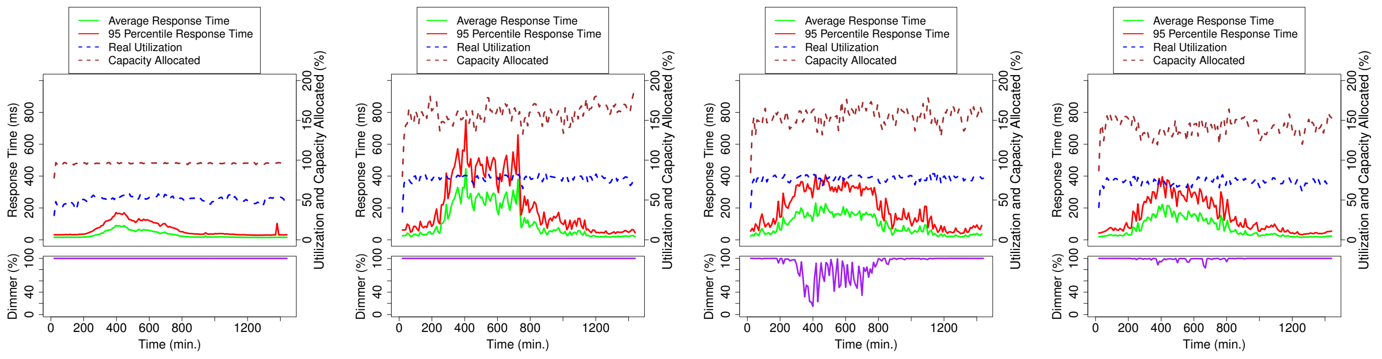
(d) Extreme varying workload

Fig. 2: Workloads for web service applications.

We aimed at keeping *95-percentile response times below 1 second*, as recommended for interactive, non-real-time applications [13], [14]. Note that, due to various uncertainties, such as caching effects and context switches, it is impossible to precisely maintain this target, while at the same time maximizing the amount of optional content served. As a compromise, we configured the brownout controller to maintain the *maximum* response time around 1 second, so as to leave a safety margin for the 95-percentile response time target.

### A. Applications and Workload

We create a representative cloud environment by mixing applications from two classes: services and jobs.



(a) Without overbooking (b) With overbooking (c) Naïve BOB (d) Smart BOB

Fig. 3: RUBiS performance for the predictable, Wikipedia-based workload.

The **service class** represents interactive applications that run for a long time (months, years) and are accessed by a varying number of users, which determines their resource requirements. We used two popular cloud benchmarks: *RUBiS* and *RUBBoS*. RUBiS [34] is an auction website benchmark modeled after eBay, whilst RUBBoS [35] is a bulleting board benchmark modeled after Slashdot. Comments and recommendations are optionally served, based on the value of the dimmer [15]. For both of them, a workload consisting of a number of queries received as a function of time is generated using information extracted from the Wikipedia [36] and FIFA [37] traces. We selected these two traces due to their complementary nature: The former has a diurnal, rather predictable trend, whereas the latter has an unpredictable trend. For each service, we selected a representative day and time-shift the original workload 12 hours for RUBiS, as shown in Fig. 2 (a) and (b). This creates different trends and peaks, to model that services may have diverse daily usage patters. Additionally, two more workloads were generated to test the system in extreme scenarios, as depicted in Fig. 2 (c) and (d), respectively. Notably, for the extreme varying workload scenario (Fig. 2 (c)) only one of the services is stressed at a time.

The **job class** consists of a realistic mix of non-interactive, batch applications with highly heterogeneous and time-varying resource requirements [21]. Their execution may take from minutes to months, during which they may present bursty or steady usage of CPU, memory, or I/O resources. We used the 3node test from the GRASP benchmark [38] (for the steady CPU behavior) and several shell scripts to generate burstiness in the different capacity dimensions. Their arrival pattern is generated using a Poisson distribution with $\lambda = 10$ seconds.

The two application classes are mixed to mimic today's data centers [39]. Initially, half of the server capacity is assigned to services, i.e., RUBiS and RUBBoS each deployed inside an 8-cores VM. The other 16 cores host jobs. Notably, due to overbooking decisions, jobs may end up utilizing more than half of the server capacity.

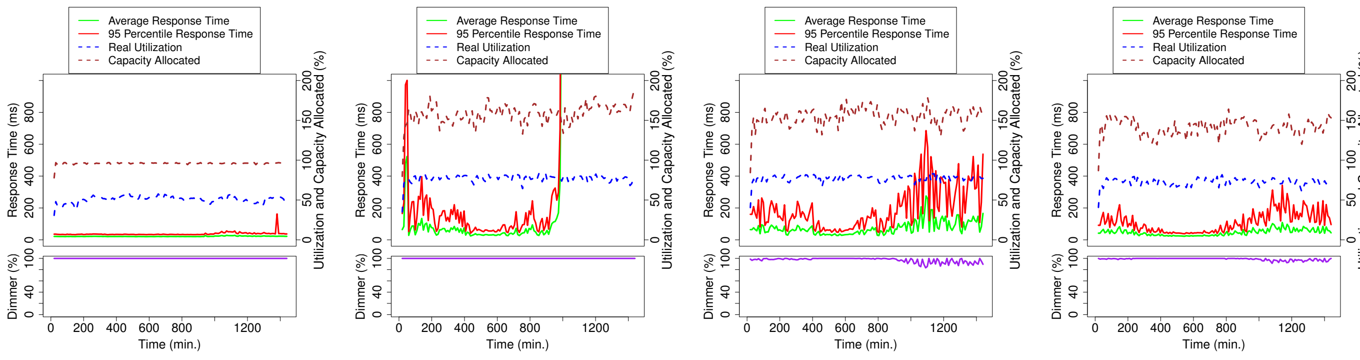
## B. Results

The evaluation is focused on services, since they are time-sensitive, hence more prone to be disturbed by overbooking decisions than jobs. We first analyze the behavior of the system when the workload trend is predictable (Wikipedia-like) and unpredictable (FIFA-like). Then, to stress-test our system, we use an extreme scenario where the load is gradually increased until the system gets fully saturated. Finally, a more varying workload scenario is used to evaluate the difference between the naïve and smart BOB approaches.

*1) Predictable Workload:* Fig. 3 shows response times and utilization levels for the RUBiS service when following the predictable workload with daily seasonality (Wikipedia). Fig. 3a and Fig. 3b show the results without and with overbooking, respectively. Overbooking (without brownout) increases real utilization and capacity allocated by roughly 50% without significant performance degradation. This is due to the fact that the overbooking framework accurately predicts the future load, which is facilitated by RUBiS's close-to-linear response time increase with the number of client queries. Even though the response times are still acceptable, they are close to the limit and a further increase in the amount of concurrent users might lead to response times exceeding the user's tolerable waiting time. Moreover, for other services the overbooking system may result in significant performance degradations, as presented for RUBBoS in Fig. 4. In that case overbooking increases the response time above 1 second, as shown in Fig. 4b from time 1000 onwards. This is caused by the fact that RUBBoS has a highly non-linear response time behavior. One may also observe a spike (not exceeding the 1 second target) at the beginning of the experiment, caused by the overbooking framework's need to learn the behavior of hosted applications[1].

By complementing overbooking with brownout, it is possible to maintain acceptable response times at all times. Fig. 3c and Fig. 3d show the improved results when the naïve and smart approaches are used with the RUBiS service. They achieve lower response times thanks to a reduction of the dimmer value (percentage of optional content being served). The same results are observed for RUBBoS services, as depicted in Fig. 4c (naïve BOB) and Fig. 4d (smart BOB), where 95-percentile response time is kept below 1 second, at the expense of dimmer reduction. Note that due to the fact that RUBBoS's optional code requires a lot of capacity, only a small dimmer reduction was necessary. Moreover, the response time spike encountered at the beginning of the experiment

[1]We have repeated this experiment several times to confirm that this is indeed a recurring phenomenon and not an experimental error.



(a) Without overbooking (b) With overbooking (c) Naïve BOB (d) Smart BOB

Fig. 4: RUBBoS performance for the predictable, Wikipedia-based workload.

without brownout was successfully mitigated with negligible impact on the dimmer value.

On the other hand, when comparing both BOB approaches, we see a significant difference in the dimmer values. The communication between the brownout controllers and overbooking framework (smart BOB) allows the system to readjust itself and slightly lower the overbooking pressure during the peak of concurrent users, this way the amount of time that the dimmer has to be reduced is significantly decreased (especially for the RUBiS service), whilst the response time are also noticeable reduced (especially for the RUBBoS service).

*2) Unpredictable Workload:* As presented in Fig. 3 and Fig. 4, when the workload is predictable and the service response time is linear to the number of queries, the overbooking framework alone is enough to keep the performance over time. However, when one or both of these conditions are not fulfilled, the overbooking system may need the help of brownout to avoid temporary performance degradation, and coordination of both techniques is required to keep the dimmer high.

As Fig. 5a illustrates, even without overbooking, the RUBBoS service is not responding properly all the time with a peak around minute 650. When overbooking framework is used (see Fig. 5b), utilization is increased around 50% but at the expense of more disturbances and response times above one second for a period of time (same as without overbooking, but wider). By using the brownout approach as a mitigation technique (Fig. 5c) the high response times around minute 650 are avoided thanks to dimmer reduction. Finally, smart BOB (Fig. 5d) further reduces the response times and the dimmer is used less often. Therefore, the smart BOB proposal adapts itself to the current needs over time, achieving higher utilization rates when services allow so, and lower rates upon unexpected situations or when services are more disturbed by overbooking decisions.

Plots regarding RUBiS service following FIFA-based workload are omitted as they do not present any interesting results — in all the cases the response times are kept below half a second without reducing the dimmer — mainly because the workload is not as high as for the Wikipedia traces even though it changes faster (Fig. 2a and Fig. 2b), and due to the more linear response time increase with the number of client queries that RUBiS presents (see Table I).

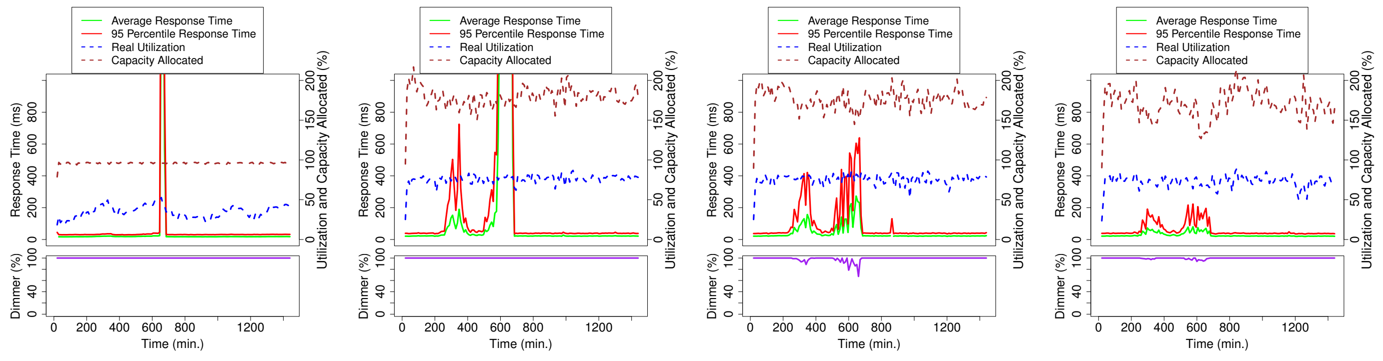
*3) Extreme Increasing Workload:* In order to test the infrastructure in a more extreme scenario, we have also tested the performance of RUBiS when the load is progressively increased until saturation, increasing the number of concurrent users by 100 every hour as presented in Fig. 2 (c).

Fig. 6 (a) to (d) show the results obtained for RUBiS without overbooking, with overbooking, naïve BOB, and smart BOB, respectively. From those plots three main advantages of smart BOB over naïve BOB can be observed:

1) Response times are better maintained (around half a second after the saturation). Note that this metric is improved even when compared to the non-overbooking scenario.
2) The level of overbooking is dynamically adjusted over time (capacity allocated) depending on the performance of the service.
3) The dimmer value remains higher.

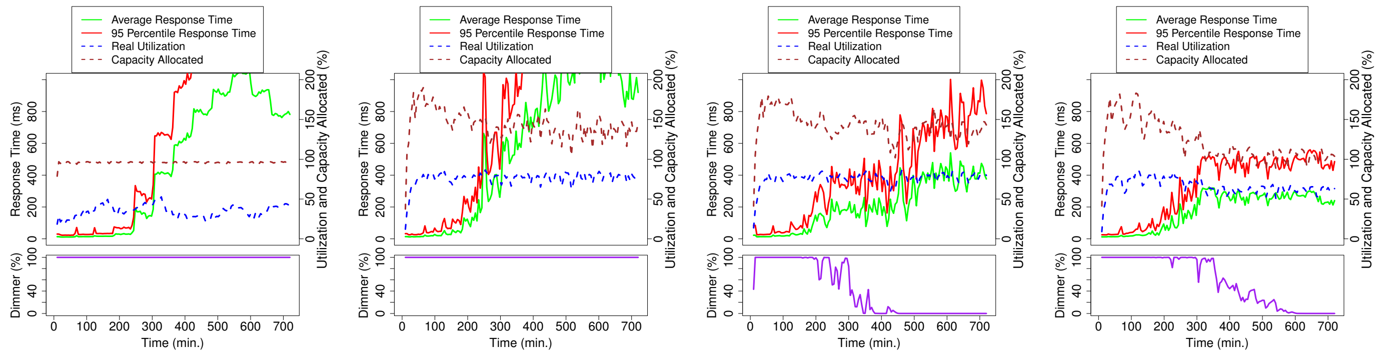
As Fig. 6d shows, the level of overbooking (capacity allocated) is around 150% of the real capacity during the first 3 hours and then starts decreasing to keep the performance of the accepted applications. The overbooking is reduced up to the point of almost having no overbooking at all (around 100% capacity allocated from Minute 300 onwards). From that point, the system is too saturated (even without overbooking) and the service needs to reduce the dimmer provided. For the naïve BOB no recommendations are provided at all from Minute 450 whilst for smart BOB this occurs from minute 600. For the cases where brownout is not used (Fig. 6a and Fig. 6b), from time 600 onwards, around 20% user queries are not served due to saturation. Therefore, if we compare those situations with the smart BOB results, we obtain an increased resource utilization, with lower response times and at least serving all the user queries even though they are served without optional content for the most saturated time interval.

Table I shows a summary of all the tests previously presented. To sum up, the smart BOB approach keeps the overbooking and utilization levels roughly the same as the other overbooking techniques but reduces the response time of the allocated services remarkably. This effect is most prominent for the services where the response time does not increase linearly with the number of users or when the number of concurrent users varies unpredictably. This fact is highlighted



(a) Without overbooking (b) With overbooking (c) Naïve BOB (d) Smart BOB

Fig. 5: RUBBoS performance for the FIFA-based workload.



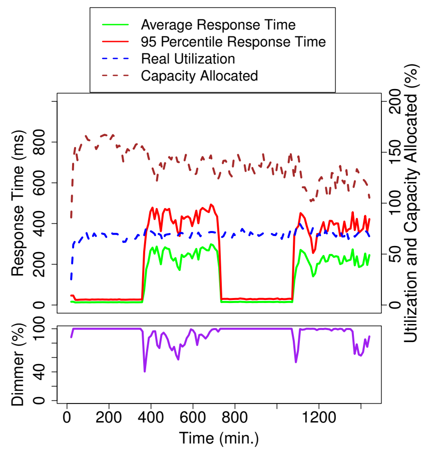
(a) Without overbooking (b) With overbooking (c) Naïve BOB (d) Smart BOB

Fig. 6: RUBiS performance for the extreme incremental workload.

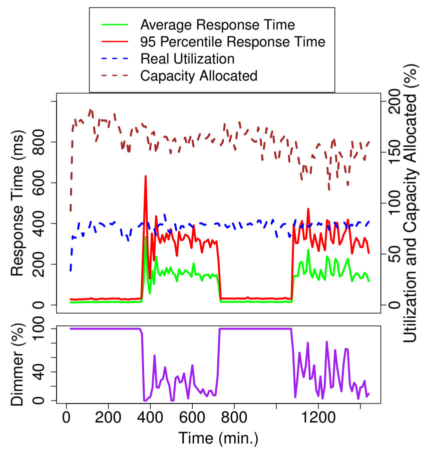
TABLE I: Summary of the tests. Values in bold highlight deficiencies of a given technique.

| Workload | Service | Technique | Response Time (ms) | | Utilization (%) | Dimmer Value (%) |
|---|---|---|---|---|---|---|
| | | | Average | Max. of 95th | | |
| Wiki | RUBiS (min. 200-800) (Fig. 3) | No Over | 51 | 171 | **53** | 100 |
| | | Over | 218 | 752 | 78 | 100 |
| | | Naïve BOB | 157 | 408 | 78 | **71.4** |
| | | Smart BOB | 138 | 396 | 72 | 98.1 |
| | RUBBoS (min. 800-1400) (Fig. 4) | No Over | 23 | 162 | **50** | 100 |
| | | Over | 1458 | **4166** | 78 | 100 |
| | | Naïve BOB | 99 | 685 | 78 | 94.6 |
| | | Smart BOB | 59 | 343 | 73 | 97.5 |
| FIFA | RUBiS | No Over | 10 | 22 | **35** | 100 |
| | | Over | 21 | 420 | 76 | 100 |
| | | Naïve BOB | 20 | 287 | 76 | 99.6 |
| | | Smart BOB | 23 | 42 | 73 | 99.6 |
| | RUBBoS (min. 200-700) (Fig. 5) | No Over | 103 | **3522** | **40** | 100 |
| | | Over | 415 | **4479** | 76 | 100 |
| | | Naïve BOB | 78 | 639 | 77 | 96.5 |
| | | Smart BOB | 42 | 223 | 72 | 99.0 |
| Extreme | RUBiS (Fig. 6) | NoOver | 477 | **1640** | **55** | 100 |
| | | Over | 620 | **2852** | 76 | 100 |
| | | Naïve BOB | 210 | 1001 | 76 | **40.5** (**25.3** in interval 200-600) |
| | | Smart BOB | 182 | 560 | 66 | 57.5 (53.8 in interval 200-600) |

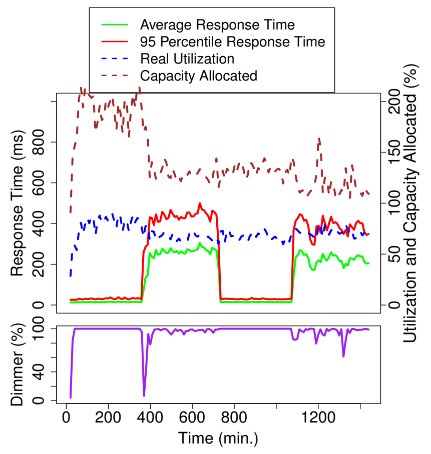
for the RUBBoS service under the FIFA traces load, where not only the level of utilization is increased compared to the non-overbooking technique, but the response time is also reduced — 32.9% average response time reduction with 32 percentage-points increment on resource utilization. In the extreme scenario it can be seen that the dimmer is maintained higher in the smart BOB approach, 28.5 percentage-points more on average for the interesting time period, i.e., where



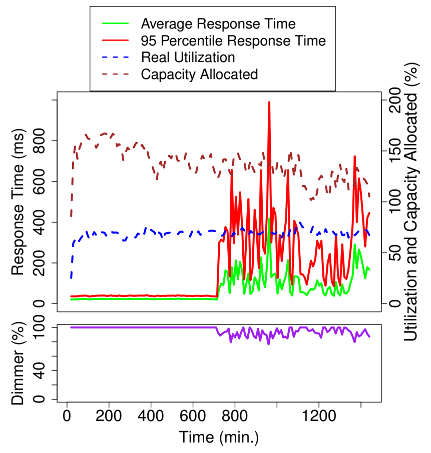
(a) RUBiS with naïve BOB (target utilization 70%)



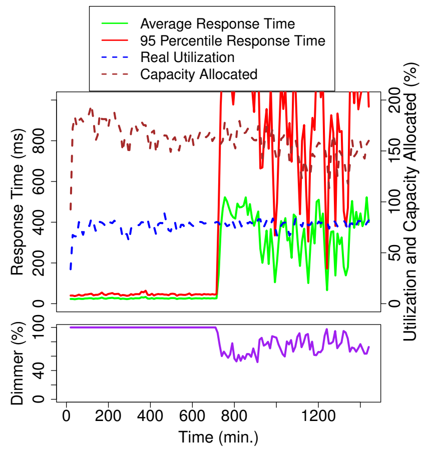
(b) RUBiS with naïve BOB (target utilization 80%)



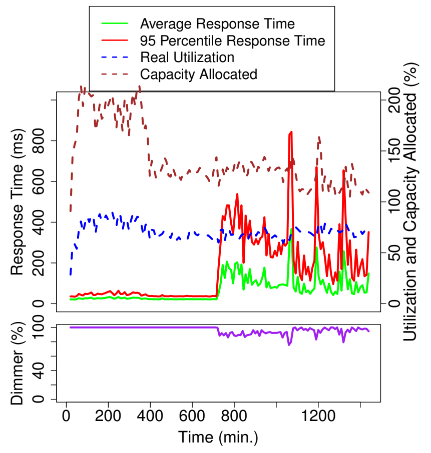
(c) RUBiS with smart BOB



(d) RUBBoS with naïve BOB (target utilization 70%)



(e) RUBBoS with naïve BOB (target utilization 80%)



(f) RUBBoS with smart BOB

Fig. 7: RUBiS and RUBBoS performance for varying workload.

the dimmer is progressively reduced from 100 to 0, while the average dimmer increment over the whole experiment is 17 percentage-points.

*4) Extreme Varying Workload:* In order to compare the performance provided by naïve and smart BOB under different situations, a test was performed where RUBiS and RUBBoS workloads change over time, showing 4 different phases (see Fig. 2d). The main objective of this test was to investigate how smart BOB self-optimizes to the different situations and applications needs over time.

Fig. 7 shows the results for RUBiS and RUBBoS services following the workload depicted in Fig. 2d for the naïve and smart BOB approaches. For the naïve BOB we have tried two different configurations, one more pessimistic, only pursuing a 70% utilization level (Fig. 7a and Fig. 7d), and one more optimistic, targeting 80% utilization rate (Fig. 7b and Fig. 7e). For the smart BOB, the target utilization is allowed to range between 60% and 85%, depending on the applications' performance (Fig. 7c and Fig. 7f). Additionally, a summary of these figures is outlined in Table II, where results per phase and aggreated numbers are presented.

When comparing naïve BOB pursuing 70% utilization with smart BOB, there are no significant differences regarding response times, both in average and 95%. However, overall and for highly loaded periods, both utilization and dimmer values are slightly higher for smart BOB. Moreover, if we take a look at the less loaded period of time (interval from minute 0 to 360), utilization is increased by 12.1 percentage-points when the smart BOB approach is used.

If naïve BOB is configured to target higher utilization levels, 80% in this case, then the overall utilization (and the one obtained at the most loaded periods) is higher than for the smart BOB approach. However, for the RUBiS service this leads to a large reduction of the dimmer value over time in order to keep the response times – unlike with smart BOB, where there is no need to reduce the dimmer value noticeably. Moreover, for the RUBBoS service, the dimmer reduction is also higher and the response times (both average and 95%) are significantly higher for the naïve approach. Average dimmer values for high loaded intervals are up to 75 and 22 percentage-points higher for smart BOB for RUBiS and RUBBoS services, respectively.

Finally, if these two approaches are compared for the time interval when the services are less loaded, we obtain similar

TABLE II: Summary of the varying workload test. Values in bold highlight deficiencies of a given technique.

| Service | Technique | Time interval (s) | Response Time (ms) | | Utilization (%) | Dimmer Value (%) |
|---|---|---|---|---|---|---|
| | | | Average | Max. of 95th | | |
| RUBiS | Naïve BOB (70%) | T1 (1-360) | 15.3 | 261.3 | **65.6** | 99.5 |
| | | T1 (361-720) | 249.0 | 493.4 | 69.8 | **84.6** |
| | | T3 (721-1080) | 17.9 | 343.3 | 69.2 | 99.5 |
| | | T4 (1081-1440) | 219.1 | 455.8 | 71.1 | 91.4 |
| | | Overall | 128.5 | 493.4 | 68.9 | 93.7 |
| | Naïve BOB (80%) | T1 | 17.1 | 328.7 | 73.9 | 99.8 |
| | | T2 | 155.0 | 633.0 | 77.9 | **19.7** |
| | | T3 | 19.1 | 396.9 | 77.4 | 99.1 |
| | | T4 | 165.6 | 472.5 | 79.1 | **32.8** |
| | | Overall | 91.4 | 633.0 | 77.1 | **61.8** |
| | Smart BOB | T1 | 16.7 | 256.9 | 77.7 | 99.6 |
| | | T2 | 255.5 | 500.5 | 67.5 | 92.8 |
| | | T3 | 18.1 | 367.2 | 67.4 | 99.6 |
| | | T4 | 224.9 | 445.8 | 71.1 | 95.1 |
| | | Overall | 131.9 | 500.5 | 70.9 | 96.7 |
| RUBBoS | Naïve BOB (70%) | T1 | 22.7 | 41.1 | **65.6** | 100 |
| | | T2 | 24.4 | 300.8 | 69.8 | 99.8 |
| | | T3 | 128.8 | 989.8 | 69.2 | 90.8 |
| | | T4 | 102.8 | 723.6 | 71.1 | 94.6 |
| | | Overall | 71.0 | 989.8 | 68.9 | 96.2 |
| | Naïve BOB (80%) | T1 | 25.7 | 58.6 | **73.9** | 100 |
| | | T2 | 29.4 | 456.4 | 77.9 | 99.8 |
| | | T3 | 381.4 | **1275.3** | 77.4 | **68.8** |
| | | T4 | 329.9 | **1249.9** | 79.1 | **76.8** |
| | | Overall | 196.4 | **1275.3** | 77.1 | 85.9 |
| | Smart BOB | T1 | 26.7 | 63.7 | 77.7 | 100 |
| | | T2 | 23.0 | 177.7 | 67.5 | 100 |
| | | T3 | 133.0 | 843.8 | 67.4 | 90.8 |
| | | T4 | 91.2 | 670.6 | 71.1 | 96.3 |
| | | Overall | 69.8 | 843.8 | 70.9 | 96.7 |

response times and dimmer values, but the utilization is higher for smart BOB as this technique detects when the applications are behaving properly and the overbooking pressure is slightly increased — utilization rise around 4 percentage-points.

To sum up, adding an explicit communication mechanism between the application and the infrastructure, as smart BOB does over naïve BOB, improves overbooking decisions. Brownout applications are less subjected to capacity shortages, hence, they require a smaller reduction in user experience to maintain target response times.

## V. Concluding Remarks

Overbooking has proven to increase data center utilization. However, it may impact applications performance, in addition to other unexpected events, such as flash crowds or resource failures that may aggravate the situation. Therefore, not only the overbooking system needs mechanisms to react to those situations, but the application itself should also be designed in a robust way so that it can quickly react to resource shortages and avoid saturation. We present two approaches where the overbooking system and the brownout applications interact implicitly (naïve BOB) or explicitly (smart BOB) to achieve higher utilization levels with low performance degradation. Naïve BOB does achieve some benefits, hence allowing our solution to be deployed incrementally or partially, i.e., brownout can be used inside a non-supporting data center. However, full benefits can only be obtained using smart BOB. Thanks to its explicit communication, the overbooking controller reduces overbooking pressure when informed that applications struggle to keep target performance. In the meantime, the affected applications temporarily reduce user experience to avoid saturation thanks to the brownout controller. The proposed techniques increase the overall utilization around 50%, while keeping the response times below 1 second, and with negligible impact on users' experience.

As guidelines for future work, we plan on investigating how to devise differentiated pricing depending on how frequently an application is willing (or able) to tolerate temporary capacity shortages. Also, we currently rely on the hypervisor to map VM cores to physical cores. We aim at using the knowledge acquired by the overbooking controller, such as class of application (normal service, brownout service or job), to pin VM cores to physical cores so as to further reduce the impact of overbooking on application performance.

## Acknowledgments

This work was supported in part by the Swedish Research Council under grant number 2012-5908 for the Cloud Control project, and by European Commission project VISION Cloud under grant agreement no. 257019.

## References

[1] D. Gmach, J. Rolia, and L. Cherkasova, "Selling t-shirts and time shares in the cloud," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, 2012, pp. 539–546.

[2] R. Birke, A. Podzimek, L. Y. Chen, and E. Smirni, "State-of-the-practice in data center virtualization: Toward a better understanding of vm usage," in *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–12.

[3] Oracle, *Oracle Enterprise Manager Cloud Control Advanced Installation and Configuration Guide*, 2013, available online: http://docs.oracle.com/html/E24089_21/sizing.htm, visited 2014-02-19.

[4] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[5] L. Tomás and J. Tordsson, "Improving cloud infrastructure utilization through overbooking," in *ACM Cloud and Autonomic Computing Conference (CAC)*, 2013.

[6] T. Wo, Q. Sun, B. Li, and C. Hu, "Overbooking-based resource allocation in virtualized data center," in *15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, 2012, pp. 142–149.

[7] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *1st ACM Symposium on Cloud Computing (SoCC)*, 2010, pp. 241–252.

[8] Z. Guo, S. McDirmid, M. Yang, L. Zhuang, P. Zhang, Y. Luo, T. Bergan, P. Bodik, M. Musuvathi, Z. Zhang, and L. Zhou, "Failure recovery: When the cure is worse than the disease," in *14th USENIX Conference on Hot Topics in Operating Systems (HotOS)*, 2013, pp. 8–8.

[9] M. Nagappan, A. Peeler, and M. Vouk, "Modeling cloud failure data: A case study of the virtual computing lab," in *2nd Intl. Workshop on Software Engineering for Cloud Computing*, 2011, pp. 8–14.

[10] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *10th ACM International Conference on Autonomic Computing (ICAC)*, 2013, pp. 23–27.

[11] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed. Morgan & Claypool, 2013.

[12] J. Hamilton, "On designing and deploying internet-scale services," in *21st Large Installation System Administration Conference (LISA)*, 2007, pp. 18:1–18:12.

[13] F. F.-H. Nah, "A study on tolerable waiting time: how long are web users willing to wait?" *Behaviour and Information Technology*, vol. 23, no. 3, pp. 153–163, 2004.

[14] R. Shea, J. Liu, E.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.

[15] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *36th International Conference on Software Engineering*. ACM, 2014.

[16] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. ISTC–CC–TR–12–101, Apr. 2012, http://www.istc-cc.cmu.edu/publications/papers/2012/ISTC-CC-TR-12-101.pdf.

[17] R. Ghosh and V. K. Naik, "Biting off safely more than you can chew: Predictive analytics for resource over-commit in iaas cloud," in *5th Intl. Conference on Cloud Computing (CLOUD)*, 2012, pp. 25–32.

[18] L. Lu, H. Zhang, E. Smirni, G. Jiang, and K. Yoshihira, "Predictive VM consolidation on multiple resources: Beyond load balancing," in *IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, 2013, pp. 1–10.

[19] A.-F. Antonescu, P. Robinson, and T. Braun, "Dynamic SLA management with forecasting using multi-objective optimization," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2013, pp. 457–463.

[20] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira, "SLA-aware resource over-commit in an IaaS cloud," in *Conference on Network and Service Management (CNSM)*, 2012, pp. 73–81.

[21] L. Tomás and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Transactions on Cloud Computing*, vol. PrePrint, 2014.

[22] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[23] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366–1379, 2013.

[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

[25] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *ACM Cloud and Autonomic Computing Conference (CAC)*, 2013.

[26] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," in *7th ACM SIGPLAN/SIGOPS Intl. Conference on Virtual Execution Environments (VEE)*, 2011, pp. 111–120.

[27] T. F. Abdelzaher and N. Bhatti, "Web content adaptation to improve server overload behavior," in *8th Intl. conference on World Wide Web (WWW)*, 1999, pp. 1563–1577.

[28] J. Philippe, N. De Palma, F. Boyer, and O. Gruber, "Self-adaptation of service level in distributed systems," *Software Practice and Experience*, vol. 40, no. 3, pp. 259–283, Mar. 2010.

[29] Y. He, Z. Ye, Q. Fu, and S. Elnikety, "Budget-based control for interactive services with adaptive execution," in *ACM International Conference on Autonomic Computing (ICAC)*, 2012, pp. 105–114.

[30] J. Kim, S. Elnikety, Y. He, S.-W. Hwang, and S. Ren, "Qaco: Exploiting partial execution in web servers," in *ACM Cloud and Autonomic Computing Conference (CAC)*, 2013.

[31] CPU time accounting, Web page at http://www.ibm.com/developerworks/linux/linux390/perf /tuning_cputimes.html, Visited 2014-01-27.

[32] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Resource management for service level aware cloud applications," in *2nd International Workshop on Real-Time and Distributed Computing in Emerging Applications*, 2013.

[33] Service-level Aware Cloud Resource Manager, Web page at https://github.com/cristiklein/cloudish, Visited 2014-01-27.

[34] RUBiS: Rice University Bidding System, Web page at http://rubis.ow2.org/, Visited 2013-11-4.

[35] RUBBoS: Bulletin Board Benchmark, Web page at http://jmob.ow2.org/rubbos.html, Visited 2013-11-4.

[36] Page view statistics for Wikimedia projects, Web page at http://dumps.wikimedia.org/other/pagecounts-raw/, Visited 2014-01-27.

[37] 1998 World Cup Web Site Access Logs - The Internet Traffic Archive, Web page at http://ita.ee.lbl.gov/html/contrib/WorldCup.html, Visited 2014-01-27.

[38] G. Chun, H. Dail, H. Casanova, and A. Snavely, "Benchmark probes for Grid assessment," in *Proc. of 18th Intl. Parallel and Distributed Processing Symposium*, 2004, pp. 26–30.

[39] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *8th ACM European Conference on Computer Systems (EuroSys)*, 2013, pp. 351–364.