Control-Based Load-Balancing Techniques: Analysis and Performance Evaluation via a Randomized Optimization Approach[☆]

Alessandro Vittorio Papadopoulos[a,*], Cristian Klein[b], Martina Maggio[a], Jonas Dürango[a], Manfred Dellkrantz[a], Francisco Hernández-Rodriguez[b], Erik Elmroth[b], Karl-Erik Årzén[a]

[a] *Department of Automatic Control, Lund University, Lund, Sweden*
[b] *Umeå University, Umeå, Sweden*

Abstract

Cloud applications are often subject to unexpected events like flashcrowds and hardware failures. Users that expect a predictable behavior may abandon an unresponsive application when these events occur. Researchers and engineers addressed this problem on two separate fronts: first, they introduced replicas – copies of the application with the same functionality – for redundancy and scalability; second, they added a self-adaptive feature called *brownout* inside cloud applications to bound response times by modulating user experience. The presence of multiple replicas requires a dedicated component to direct incoming traffic: a load-balancer.

Existing load-balancing strategies based on response times interfere with the response time controller developed for brownout-compliant applications. In fact, the brownout approach bounds response times using a control action. Hence, the response time, that was used to aid load-balancing decision, is not a good indicator of how well a replica is performing.

[☆] This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures", and through the LCCC Linnaeus and ELLIIT Excellence Centers.

[*] Corresponding author

Email addresses: alessandro.papadopoulos@control.lth.se (Alessandro Vittorio Papadopoulos), cklein@cs.umu.se (Cristian Klein), martina.maggio@control.lth.se (Martina Maggio), jonas@control.lth.se (Jonas Dürango), manfred@control.lth.se (Manfred Dellkrantz), hernandf@cs.umu.se (Francisco Hernández-Rodriguez), elmroth@cs.umu.se (Erik Elmroth), karlerik@control.lth.se (Karl-Erik Årzén)

To fix this issue, this paper reviews some proposal for brownout-aware load-balancing and provides a comprehensive experimental evaluation that compares them. To provide formal guarantees on the load-balancing performance, we use a randomized optimization approach and apply the scenario theory. We perform an extensive set of experiments on a real machine, extending the popular lighttpd web server and load-balancer, and obtaining a production-ready implementation. Experimental results show an improvement of the user experience over Shortest Queue First (SQF) — believed to be near-optimal in the non-adaptive case. The improved user experience is obtained preserving the response time predictability.

Keywords: Load-balancing, randomized optimization, cloud control.

1. Introduction

Cloud computing has dramatically changed the management of computing infrastructures. On one hand, public infrastructure providers, such as Amazon EC2, allow service providers, such as Dropbox and Netflix, to deploy their services on large infrastructures with no upfront cost [1], by simply leasing computing capacity in the form of Virtual Machines (VMs). On the other hand, the flexibility offered by cloud technologies, which allow VMs to be hosted by any Physical Machine (PM) (or server), favors the adoption of private clouds [2]. Therefore, also self-hosting service providers are converting their computing infrastructures into small clouds.

However, the scalability of these infrastructure is a serious concern and due to their ever-increasing complexity, and hardware failures are rather common. In fact, in cloud computing infrastructures failures are the norm rather than an exception [3, 4]. This is why internet-scale interactive applications, such as e-commerce websites, include replication early in their design [5]. Replication improves scalability, i.e., more users can be served by adding more replicas, but also makes the application more resilient to failures. In case a replica fails, other replicas can take over.

In a replicated setup, a load-balancer is responsible for monitoring replicas' health and directing requests as appropriate. Indeed, this practice is well established and using it, applications can successfully deal with failures as long as the remaining computing

capacity is sufficient [5].

However, failures in cloud infrastructures are often correlated in time and space [6, 7]. Therefore, it may be economically inefficient for the service provider to provision enough spare capacity for dealing with all failures in a satisfactory manner. In case correlated failures occur, the service may *saturate*, i.e., it may no longer be able to serve users in a timely manner. This in turn leads to dissatisfied users, that can abandon the service. Note that the saturated service causes infrastructure overload, which by itself may trigger additional failures [8], aggravating the initial situation. This strongly motivates the need for a solution to deal with rare, and cascading failures, that produce temporary capacity shortages on the underlying architecture.

A promising self-adaptation technique that solves this problem is *brownout* [9, 10]. In essence, a service is extended to serve requests in two modes: with mandatory content only, such as product description in an e-commerce website, and with both mandatory and optional content, such as the product information and recommendations of similar sales. Serving more requests with optional content, increases the revenue of the provider up to 50% [11], but also greatly affects the capacity requirements of the service. The tradeoff between the two is a matter of quality of service, and it has been explored using control theory, where a controller was used to decide the percentage of requests to be served with optional content enabled [9, 10], in order to keep the response time below the user's tolerable waiting time [12]. These studies were conducted in a single replica case and subsequently extended to the multiple replica, by adding a load-balancer [13, 14].

Despite the fact that load-balancing techniques have been widely studied [3, 15, 16, 17, 18, 19, 20], state-of-the-art load-balancers forward requests based on metrics that cannot discriminate between a replica that is avoiding overload by not executing the optional code and a replica that is not subject to overload. This called for an analysis on the introduction of brownout-aware load-balancers [13]. The promising results of the analysis encouraged the implementation of brownout-aware load-balancers [14]. The implementation was compared with the state-of-the-art load-balancing algorithm that obtained the best performance in the simulation campaign, Shortest Queue First (SQF) [21].

Despite obtaining promising simulation results [13] and good implementation performance [14], the load-balancers were tested only in specific scenarios and no guarantees on the generality of the results were given. Moreover, since the events that can happen at the data center level are unpredictable and unknown, it is impossible to construct a set of scenarios that would make the validation extensive enough for a production environment. To address the problem of providing formal guarantees despite unpredictability, this paper applies the *scenario theory* [22, 23, 24] to obtain formal probabilistic guarantees about the behavior of the load-balancing strategies. The scenario theory has been recently used in many different situations especially when dealing with stochastic or uncertain systems [25, 26, 27], also in combination with robust optimization [28]. The application of the scenario theory allows one to extensively evaluate the performance of different techniques, providing some formal guarantees via the solution of a chance-constrained optimization problem.

In essence, the contribution of this paper is to provide probabilistic guarantees on the behavior of brownout-aware load-balancers and on the improvements they bring in comparison to brownout-unaware ones. The solution of this problem greatly extends the contribution of [13], that was only proposing time-based algorithms and evaluting them with a simulator, and of [14], that only used specific scenarios as case studies for the evaluation, therefore not being able to derive any guarantee on the generic behavior of the proposed load-balancers. Solving this problem required an extensive experimental campaign – this paper considers about 800 experiments for each load-balancing strategy against the 30 considered before. Also, the evaluation campaign conducted for this extension considers a more realistic experimental setting – both the workload characterization and the resource availability are here randomized. The adoption of the scenario theory and the required additional data provide probabilistic guarantees on the worst-case obtainable performance – these guarantees can not be obtained with the bare statistical analysis conducted in the past.

The solution proposed in this paper can be applied to a wider class of applications. Indeed, in self-adaptive software systems, there is a strong need for a different type of evaluation campaign, where all the sources or randomness in the execution are properly identified and all the possible configurations are potentially tested. This paper presents

the first attempt of using the scenario theory to provide this analysis.

Results show that the cloud application can tolerate more replica failures and that the novel load-balancing algorithms improve the number of requests served with optional content, by up to 5%, the previous results of [14] on specific cases. In addition, the worst case scenario is improved by 15% with respect to SQF, with a very high probability that is quantified through the application of the scenario theory. This result is especially remarkable, since SQF is believed to be near-optimal, in the sense that it minimizes the average response time for non-brownout-enabled replicas [21, 29].

The rest of the paper is organized as follows. Section 2 discusses the state of the art, while Section 3 describes in more detail the brownout framework, better highlighting the contribution of this paper. Section 4 details the proposed control-theoretical load-balancers and some of their implementation aspects. Section 5 introduces the scenario theory and its application to test the different load-balancing strategies, highlighting the obtainable formal guarantees. Section 6 presents an extensive set of experiments that are evaluated with the scenario theory, and compared with previous results. Section 7 concludes the paper and sketches possible future work.

To make our results reproducible and foster further research on improved resilience through brownout, the source code for the load-balancer and to run the experiments has been released online[1].

2. Related work

Load-balancers are standard components of internet-scale services [30], allowing applications to achieve scalability and resilience [3, 5, 31, 32]. Many load-balancing policies have been proposed, aiming at different optimizations, spanning from equalizing processor load [33] to managing memory pools [34, 35], to specific optimizations for iterative algorithms [36]. Typically, the load-balancer assumes the number of available resources to be constant since this aspect is managed at a larger time scale — usually in the time scale of tens of minutes. On the other hand, the *autoscaler* is in

[1] `https://github.com/cloud-control/brownout-lb-lighttpd`

charge of adapting the number of available resources according to the incoming workload [37]. The choice of the autoscaler is critical for many different reasons, and, in particular, for pricing issues, like for example, for the minimization of power consumption in a data center [38, 37]. In the following, we assume that an autoscaler is in place, and that the number of incoming requests is not changing the number of available resources, therefore we focus only on the load-balancing algorithm.

Often load-balancing policies consider web server systems as a target [39, 40], where one of the most important goal is to bound the maximum response time that the clients are exposed to [41]. Load-balancing strategies can be guided by many different purposes, for example geographical [42, 43], driven by the electricity price to reduce the datacenter operation cost [44], or specifically designed for cloud applications [3, 15, 16].

Load-balancing solutions can be divided into two different types: static and dynamic. Static load-balancing refers to a fixed strategy to route traffic [45, 46]. The most commonly used technique is based on selecting each replica in turn, called Round Robin (RR). It can be either deterministic, storing the last selected replica, or probabilistic, picking one at random. However, due to their static nature, such techniques would not have good performance when applied to brownout-compliant applications. A static policy would not take into account the inherent fluctuations of a cloud environment, like bursty workloads, the sudden popularity of a page, a service or even a whole application. Moreover, static techniques disregard the control strategy at the replica level, which leads to changing capabilities of replicas [13]. Our contribution is designed to deal with failures reactively. Failure prediction [4], if accurate enough, could be used to improve the control strategy, but it is usually difficult to achieve in a real-time environment.

On the contrary, dynamic load-balancing is based on measurements of the current system's state. One popular option is to choose the replica which had the lowest response time in the past. We refer to this algorithm as Fastest Replica First (FRF) if the choice is based on the last measured response time of each replica, and FRF-EWMA if the choice is based on an Exponentially Weighted Moving Average over the past response times of each replica. A variation of this algorithm is Two Random Choices

(2RC) [47], that randomly chooses two replicas and assigns the request to the fastest one, i.e., the one with the lowest maximum response time. Through simulation results, we were able to determine that FRF, FRF-EWMA and 2RC are unsuitable for brownout applications [13].

Another adopted strategy is based on the pending request count and generally called SQF, where the load-balancer tracks the pending requests and selects the replicas with the least number of requests waiting for completion. This strategy has proven to be close to optimal for non-adaptive replicas [21, 29] and pays off especially in architectures where the replicas have similar capacities and the requests are homogeneous. To account for non-homogeneity, Pao and Chen proposed a load-balancing solution using the remaining capacity of the replicas to determine how the next request should be managed [48]. The capacity is determined through a combination of factors like the remaining available CPU and memory, the network transmission and the current pending request count. Other approaches have been proposed that base their decision on remaining capacity. However, due to the fact that brownout applications indirectly control CPU utilization, by adjusting the execution of optional content, so as to prepare for possible request bursts, deciding on remaining capacity alone is not an indicator of how a brownout replica is performing.

A merge of the fastest replica and the pending request count approach was implemented in the BIG-IP Local Traffic Manager [49], where the replicas are ranked based on a linear combination of response times and number of routed requests. Since the exact specification of this algorithm is not open, we tried to mimic this behavior in simulation [13], according to the description provided in [49]. One of the solutions proposed in this paper extends the idea of looking at the difference between the past behavior and the current one, although the metric observed by our solution is the ratio of optional content served, which is used to decide how to route traffic so as to maximize optional content served.

Dynamic solutions can be control-theoretical [50, 51] and also account for the cost of applying the control action [52] or for the load trend [53]. This is especially necessary when the load-balancer also acts as a resource allocator deciding not only where to route the current request but also how much resources it would have to allocate to

a replica, like in [54]. In these cases, the induced sudden lack of resources can result in poor performance. However, we focus only on load-balancing solutions, since brownout applications are already taking care of the potential lack of resources [9].

The following section reviews the necessary background about brownout applications and SQF, which is used as a comparison point. Our previous simulation results [13] show that SQF is the only non-brownout-aware load-balancing algorithm obtaining a competitive performance in terms of optional content served, and it is therefore the only one used in the following experimental evaluation, as a comparison point.

3. Background and Motivation

3.1. Single Replica Brownout Services

To provide predictable performance in cloud services, the brownout paradigm [9, 10] relies on a few, minimally intrusive code changes and a control strategy for the response time of a single-replica based service. The service programmer builds a brownout-compliant cloud service breaking the service code into two distinct subsets: Some functions are marked as *mandatory*, while others as *optional*. For example, in an e-commerce website, retrieving the characteristics of a product from the database can be seen as mandatory – a user would not consider the response useful without this information – while obtaining reviews and recommendations of similar products can be seen as optional – this information enhances the quality of experience of the user, but the response is useful also without them. In the following we consider the incoming requests to be homogeneous, i.e., the type of requests is the same, as in the case of an e-commerce website like Amazon. The case of heterogeneous requests is left as a future work.

For a brownout-compliant service, whenever a request is received, the mandatory part of the response is always computed, whereas the optional part of the response is produced only with a certain probability given by a control variable, called the dimmer value. Not executing the optional code reduces the computing capacity requirements of the service, but also degrades user experience. Clearly, the user would have more information by seeing optional content, such as related products and comments from

other users. However, in case of overload and transient failure conditions, it is better to obtain partial information than to have increased response times or no response, due to insufficient capacity.

Keeping the service responsive is done by adjusting the probability of executing the optional components [9]. Specifically, a controller monitors response times and adjusts the dimmer value to keep the 95th percentile response time observed by the users around a certain setpoint. Focusing on the 95th percentile instead of the average value, allows more users to receive a timely response, hence improve their satisfaction [55]. A setpoint of one second was used, to leave a safety margin to the user's tolerable waiting time, estimated to be around four seconds [12]. While the initial purpose of brownout was to enhance the service's tolerance to a sudden increase of the workload, it also significantly improves responsiveness during infrastructure overload phases, when the service is not allocated enough capacity to manage the amount of incoming requests without degrading the user experience. However, the brownout approach was used only in services composed of a single replica, thus the service could not tolerate hardware failures.

The control design for the single replica in [9] is based on the linear model:

$$\tau(k+1) = \alpha \cdot \Theta(k), \tag{1}$$

where $\tau(\cdot)$ is the predicted 95th percentile of the response time empirical distribution, $\Theta(k)$ is the last dimmer value, while $\alpha \in \mathbb{R}$ is a parameter that is in general unknown. Admittedly, the model is very simplistic, but one of the main difficulties in the computing systems domain is that there is no physics that helps one to design equation-based models. Therefore, in [9] the fixed-structure model (1) is assumed, and an estimate $\hat{\alpha}$ of the parameter α is identified online through a Recursive Least Square (RLS) filter with a forgetting factor $f \in (0, 1]$ [10]. Then, an adaptive PI (Proportional Integral) is designed as:

$$\Theta(k+1) = \Theta(k) + \frac{1-p}{\hat{\alpha}(k)} \cdot \left(\tau^{\circ}(k) - \tau(k)\right), \tag{2}$$

where p is the only design parameter that is tuned empirically [10], $\tau^{\circ}(k)$ and $\tau(k)$ are the setpoint and measured value of the 95th percentile of the response time empirical

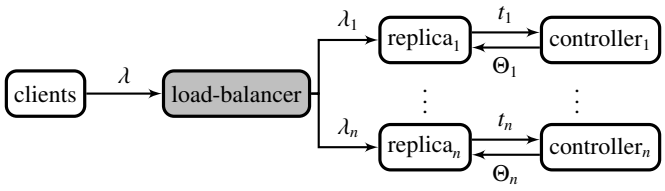

Figure 1: Architecture of a brownout cloud service featuring multiple replicas.

distribution. For these parameters, we set $\tau^{\circ}(k) = 1\text{s}$, $f = 0.95$ and $p = 0.9$.

3.2. Multiple Replica Brownout-Compliant Services

For fault tolerance and scalability reasons, cloud services should feature multiple replicas. Fig. 1 illustrates the software architecture that is deployed to execute a brownout-compliant service composed of multiple replicas. Besides the addition of replica controllers to make it brownout-compliant, the architecture is widely accepted as the reference one for replicated cloud services [3].

In the given cloud service architecture, the client requests are assumed to arrive according to a Poisson process with intensity λ, which is unknown, and possibly time-varying. Each client request is received by the load-balancer, that forwards it to one of the n replicas. Each replica independently decides if the request should be served with or without the optional part. The chosen replica produces the response and sends it back to the load-balancer, which forwards it to the original client. Since all responses of the replicas go through the load-balancer, it is possible to piggy-back replica status information to aid balancing decisions. For the purpose of this paper, we assume that each replica i piggy-backs the current value of the dimmer Θ_i through the response, so that this value is known by the load-balancer. However, to keep the system as decoupled as possible, the load-balancer does not have any knowledge on *how* each replica controller adjusts Θ_i. This decoupling opens up the path to making the load-balancer itself replicated, which is left for future work.

In the end, each replica i receives a fraction λ_i of the incoming traffic and serves requests with a 95th percentile response time around the same setpoint of 1 second. Each replica i chooses a dimmer Θ_i that depends on the amount of traffic it receives and

the computing capacity available to it. Directing too many requests to a certain replica, the load-balancer may indirectly decrease the amount of optional requests served by that replica.

Preliminary simulation results [13] compared different load-balancing algorithms for this architecture. The main result of this comparison is that load-balancing algorithms that are based on measurements of the response times of the single replicas are not suited to be used with brownout-compliant services, since the replica controllers already keep the response times close to the setpoint. The *only* existing algorithm that proved to work adequately with brownout-compliant services is Shortest Queue First (SQF). It works by tracking the number of queued requests q_i on each replica and directing the next request to the replica with the lowest q_i.

3.3. Resilience without and with Brownout

Before delving into the load-balancing strategies development, we show through experiments how brownout can increase resilience, even when used with a brownout-unaware load-balancing algorithm, such as SQF. To this end, we expose both a non-brownout and a brownout service to cascading failures and their recovery. The infrastructure of this experiment starts with 5 replicas of a web application, each being allocated 4 cores, i.e., the service is allocated a total computing capacity of 20 cores. Every 100 seconds a replica crashes until only a single one is active. Then, every 100 seconds a replica is restored. To focus on the behavior of the service due to failure, we kept the request-rate constant at 200 requests per second. Note that, each replica was configured with enough soft resources (file descriptors, sockets, etc.) to deal with 2500 simultaneous requests. The user requests are timed out if the service is not provided within the tolerable waiting time of 4 seconds [12].

Fig. 2 shows the amount of requests per second that have been timed out and the optional content ratio served.

The non-brownout service performs well even with 2 failed replicas, from 0 to 300 seconds. Indeed, there are no timeouts and all requests are served with optional content. The web server that we used in the experiments is `lighttpd`, which already includes code to retry a failing request on a different replica, hence hiding the failure from the

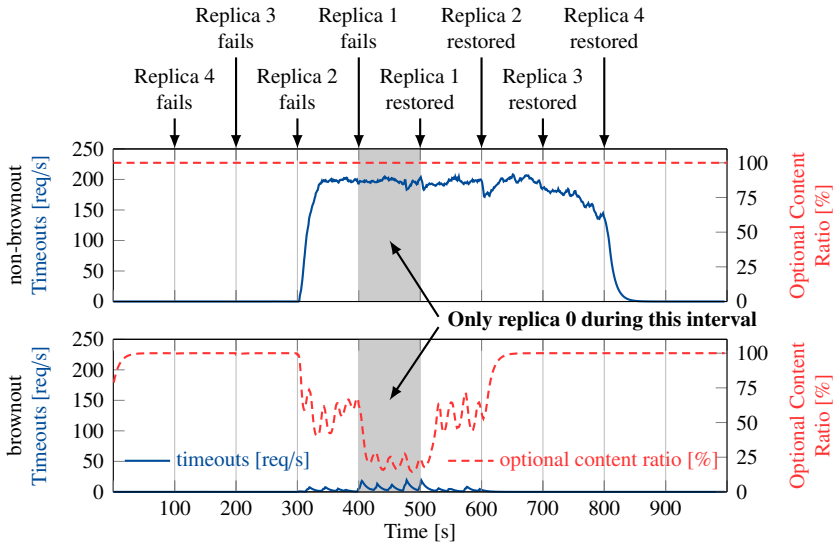

Figure 2: Experimental results comparing resilience without and with brownout. Configuration: 5 replicas, each having 4 cores.

user. During the time interval from 0 to 300 seconds, the brownout service performs almost identically, except negligible reductions in optional content ratio at start-up and when a replica fails, until the replica controller adapts to the new conditions. However, starting at time 300, when the third replica fails, the non-brownout service behaves poorly. The remaining computing capacity is insufficient to serve the incoming requests fast enough and response time starts increasing. A few seconds later the service is saturated and almost all incoming requests time out. When enough replicas are restored to make capacity sufficient, the non-brownout service still does not recover. It recovers only when all the replicas are restored.

On the other hand, the brownout service performs well even with few active replicas. At time 300, when the third replica fails leading the service into capacity insufficiency, the replica controllers detect the increase in response time and quickly reacts by reducing the optional content ratio to around 55%. As a results, the service does not saturate and users can continue enjoying a responsive service. At time 400 when the fourth replica fails, capacity available to the service is barely sufficient to serve any re-

quests, even with zero optional content ratio. However, even in this case, the brownout service significantly reduces the number of timeouts by keeping the optional content ratio low, around 10%. Finally, when replicas are restored, the service recovers fairly quickly. Thanks to the action of the replica controllers, the database servers do not fill up with "rotten" requests.

Noteworthy, qualitatively identical results can be obtained with experiments in different conditions (number of allocated cores, different realizations of the workloads, etc.). See for example the ones included in [14].

In summary, adding brownout to a replicated service improves its resilience, even when using a brownout-unaware load-balancing algorithm. The increase in resilience that can be obtained is specific to each service. In the following, we are interested in the improvements that can be obtained with load-balancers specifically designed for (brownout-aware) applications.

4. Design and Implementation

This section presents two brownout-aware load-balancing algorithms and their production-ready implementations.

4.1. Brownout-Compliant Load-Balancing Algorithms

Here we discuss two brownout-compliant control-based load-balancing algorithms. The algorithms are based on the ideas presented in [13, 14], with two major modifications. First, all the techniques strive to maximize the optional content served by acting on the fraction of incoming traffic sent to a specific replica, while here the algorithms are acting in an SQF-like way but with queue-offsets that are dynamically changed in time. The queue-offsets u_i take into account the measured performance of each replica i in terms of its dimmer value, and are subtracted from the actual value of the queue length q_i so as to send the request to the replica with the lowest $q_i(k) - u_i(k)$.

Second, in previous contributions, all the algorithms run periodically, independently of the incoming traffic, while in this paper we are considering algorithms that are fully event-driven, updating the queue-offsets and taking a decision for each request.

These two modifications improve the achieved performance, both in terms of optional content served and response time, rendering the service more reactive to sudden capacity changes, as in the case with failures. We now present two control laws for computing the queue-offsets u_i.

Even though the proposed algorithms are based on the fact that brownout-applications are running, they can be used also with non-brownout-enabled applications, just by considering $\Theta_i(k) = 1, \forall k, i$. This allows also for having hybrid environment with both brownout- and non-brownout-enabled applications.

4.1.1. PI-Based Heuristic (PIBH)

The first controller is based on a variant of the PI (Proportional and Integral) controller in velocity form. In principle, the PI control action in velocity form is based both on the variation of the dimmers value (which is related to the proportional part), and their actual values (which is related to the integral part).

As presented above, the values of the queue offsets u_i are updated every time a new request is received by the service, according to the last values of the dimmers Θ_i, piggy-backed by each replica i through a previous response, and on the queue lengths q_i, using the formula

$$u_i(k+1) = (1-\gamma)\left[u_i(k) + \gamma_P h\, \Delta\Theta_i(k) + \gamma_I h\, \Theta_i(k)\right] + \gamma h q_i(k), \tag{3}$$

where $\gamma \in (0, 1)$ is a filtering constant, γ_P and γ_I are *constant* gains related to the proportional and integral action of the classical PI controller, and h is the time elapsed from the last control intervention, i.e., from the last request. Of course, h is saturated in such a way that $1 - \gamma h \geq 0$, i.e., $h \leq 1/\gamma$. Notice that h has the same probability distribution as the incoming traffic.

We selected $\gamma = 0.01$ and $\gamma_P = 1.0$ based on empirical tuning. Once γ and γ_P are fixed to a selected value, increasing the integral gain γ_I calls for a stronger action on the load-balancing side, which means that the load-balancer would take decisions very much influenced by the current values of Θ_i, therefore greatly improving performance at the cost of a more aggressive control action. On the contrary, decreasing γ_I would smoothen the control action, possibly resulting in performance loss due to a slower

reaction time. The choice of the integral gain allows to exploit the trade-off between performance and robustness. For the experiments we set $\gamma_I = 0.5$.

4.1.2. Equality Principle-Based Heuristic (EPBH)

The second algorithm is based on the heuristic that the system will perform well in a situation when all replicas have the same dimmer value. By comparing Θ_i for each replica i with the average dimmer of all replicas, the designed update rule can deduce which replica should receive more load, in order to drive all dimmer to be equal. The queue offsets can thus be updated as

$$u_i(k+1) = u_i(k) + \gamma_e h \left(\Theta_i(k) - \frac{1}{n} \sum_{j=1}^{n} \Theta_j(k) \right), \tag{4}$$

where γ_e is a constant gain and h is the time elapsed from the last control intervention. The gain decides how fast the controller should act. Based on empirical tuning we set $\gamma_e = 0.1$.

4.2. Implementation

In order to show the practical applicability of the two controllers and evaluate their performance, we decided to implement them in an existing load-balancing software. We chose `lighttpd`[2], a popular open-source web server and load-balancing software, that features good scalability, thanks to an event-driven design. `lighttpd` already included all necessary prerequisites, such as HTTP request forwarding, HTTP response header parsing, replica failure detection and the state-of-the-art queue-length-based SQF algorithm. HTTP response header parsing allowed us to easily implement dimmer piggy-backing through the custom `X-Dimmer` HTTP response header, adding only 20 bytes of overhead. In the end, we obtained a production-ready brownout-aware load-balancer implementation featuring the two algorithms, with less than 180 source lines of C code[3].

[2] `http://www.lighttpd.net/`

[3] `https://github.com/cloud-control/brownout-lb-lighttpd`

5. Scenario Theory: A Randomized Method for Performance Evaluation

The evaluation of the presented techniques is a non-trivial task, due to the different sources of randomness that arise in a cloud environment, ranging from hardware failures to flash crowd events. It is thus necessary to specifically design a performance evaluation technique that is able to cope with such a stochastic behavior of the infrastructure.

In this section, a randomized method is described for evaluating the performance of different load-balancing policies, when the incoming traffic rate λ is stochastic and the goal is to maximize the optional content served over a finite number of requests R. The main contribution of this paper is the formulation of the performance evaluation methodology as a chance-constrained optimization problem.

The proposed method feeds the different load-balancing policies with a number of realizations of the stochastic input. We consider two different types of events that may occur in the system, i.e., the arrival rate of requests λ into the system, and the possible failures of the replicas over time.

Our goal is to introduce a method for evaluating performance of the proposed methodologies with probabilistic guarantees. The aim of the performance evaluation is to evaluate how much optional content the system is able to serve, and how much the response time $\tau \in \mathbb{R}^+$ of a request exceeds the prescribed setpoint $\tau^\circ \in \mathbb{R}^+$. Using the index k to count the requests, we denote:

$$\tau_{\text{exc}} := \max\{\tau(k) - \tau^\circ, 0\}.$$

Therefore, it is possible to define the output of the system as:

$$y(k) = \begin{bmatrix} \text{OC}(k) & \tau_{\text{exc}}(k) \end{bmatrix} \tag{5}$$

with $\text{OC}(k) \in \{0, 1\}$ being the binary variable that says if the k-th request was served or not with optional content. The desired (ideal) behavior of the system – which is not necessarily always achievable – is then $y^\circ(k) = [1\ 0]$, i.e., when the optional content is always served – since it maximizes the revenues for the service provider – and that the response time per request never exceeds the setpoint. Solutions that serve the same

amount of optional content, with a response time that never exceeds the setpoint are therefore equivalent for this study.

In order to appropriately evaluate the performance of a policy, we define a distance $d_R(\cdot,\cdot)$ that maps each pair of trajectories $y(k)$, and $y^\circ(k)$, $k \in [1, 2, \ldots, R]$, into a positive real number $d_R(y, y^\circ)$ that represents the extent to which the output y is far from the optimal behavior y° along a finite number of R incoming requests.

Note that $d_R(y, y^\circ)$ is a random quantity since it depends on the realization of the stochastic workload $w(k)$ and the stochastic realization of the mentioned quantities in the system.

The performance can be evaluated in a "worst-case" fashion, by solving the following Chance-Constrained optimization Problem (CCP):

$$\begin{aligned} CCP: \min_{\rho} \rho \\ \text{subject to: } \mathbb{P}\{d_R(y, y^\circ) \leq \rho\} \geq 1 - \epsilon. \end{aligned} \tag{6}$$

In this case, we choose the distance metric

$$\begin{aligned} d_R(y, y^\circ) =& \frac{1}{R}\|y^\circ - y\|^2 = \frac{1}{R}\left\|\begin{bmatrix} 1 - \text{OC} & 0 - \tau_{\text{exc}} \end{bmatrix}\right\|^2 \\ =& \frac{1}{R}\left\|\begin{bmatrix} 1 - \text{OC} & 0 - \max\{\tau - \tau^\circ, 0\} \end{bmatrix}\right\|^2 \\ =& \frac{1}{R}\left\|\begin{bmatrix} 1 - \text{OC} & \min\{\tau^\circ - \tau, 0\} \end{bmatrix}\right\|^2 \end{aligned} \tag{7}$$

where R is the number of requests considered in the experiment, and $\|X\|^2$ is the squared 2-norm of the matrix X. In other words, we are penalizing those policies that are serving very few optional content, and that exceed the upper bound of response time given by the setpoint τ°. Notice that methods that are serving the same amount of optional content always keeping the response time lower than the setpoint have the same distance.

Irrespectively of the choice for $d_R(y, y^\circ)$, finding the solution $\rho^\star$ of the CCP (6) is known to be an NP-hard problem [24, 56], since it involves determining, among all sets of realizations of the stochastic input and initial state that have a probability $1 - \epsilon$, the one that provides the best (lowest) value for $d_R(y, y^\circ)$. We then head for an approximate solution where instead of considering all the possible realizations for the stochastic

uncertainty, we consider only a finite number N of them called *scenarios*, randomly extracted according to their probability distribution, and treat them as if they were the only admissible uncertainty instances. The number N of scenarios corresponds to the number of experiments with different inputs to be performed for the solution of the CCP. This leads to the formulation of Algorithm 1, where the chance-constrained solution is determined using some empirical violation parameter $\eta \in (0, \epsilon)$.

Algorithm 1 Randomized solution

1: extract N realizations of the stochastic input $u^{(i)}(k)$, $k = 1, 2, \ldots, R$, $i = 1, 2, \ldots, N$, and let $\kappa = \lfloor \eta N \rfloor$, with $\eta \in (0, \epsilon)$;

2: determine the N realizations of the output signals $y^{(i)}(k)$ $k \in R$, $i = 1, 2, \ldots, N$, when the policy to be evaluated is fed by the extracted uncertainty instances;

3: compute

$$\hat{\rho}^{(i)} := d_R(y^{(i)}, y^{\circ}),\ i = 1, 2, \ldots, N;$$

4: determine the indices $\{h_1, h_2, \ldots h_\kappa\} \subset \{1, 2, \ldots, N\}$ of the κ largest values of $\{\hat{\rho}^{(i)},\ i = 1, 2, \ldots, N\}$

5: set

$$\hat{\rho}^{\star} = \max_{i \in \{1,2,\ldots,N\} \setminus \{h_1, h_2, \ldots, h_\kappa\}} \hat{\rho}^{(i)}.$$

Notably, if the number N of extractions is appropriately chosen, the obtained estimate of $\rho^{\star}$ is chance-constrained feasible, with a-priori specified (high) probability. This result is based on the *scenario theory* [26], which was first introduced for solving uncertain convex programs via randomization [22] and then extended to chance-constrained optimization problems [23].

Proposition 5.1. *Select a confidence parameter $\beta \in (0, 1)$ and an empirical violation parameter $\eta \in (0, \epsilon)$. If N is such that*

$$\sum_{i=0}^{\lfloor \eta N \rfloor} \binom{N}{i} \epsilon^i (1 - \epsilon)^{N-i} \leq \beta, \tag{8}$$

then, the solution $\hat{\rho}^\star$, to Algorithm 1 satisfies

$$\mathbb{P}\{d_R(y, y^\circ) \leq \hat{\rho}^\star\} \geq 1 - \epsilon, \tag{9}$$

with probability at least $1 - \beta$.

If we discard the confidence parameter β for a moment, this proposition states that the randomized solution $\hat{\rho}^\star$ obtained through Algorithm 1 is feasible for the CCP (6). As η tends to ϵ, $\hat{\rho}^\star$ approaches the desired optimal chance constrained solution $\rho^\star$. In turn, the computational effort grows unbounded since N scales as $1/(\epsilon - \eta)$ [23], therefore, the value for η depends in practice on the time available to perform experiments.

As for the confidence parameter β, one should note that $\hat{\rho}^\star$ is a random quantity that depends on the randomly extracted input realizations and initial conditions. It may happen that the extracted samples are not representative enough, in which case the size of the violation set will be larger than ϵ. Parameter β controls the probability that this happens and the final result holds with probability $1 - \beta$. N satisfying (8) depend logarithmically on $1/\beta$, [23], so that β can be chosen as small as 10^{-10} (yielding $1 - \beta$ be practically 1) without growing N significantly.

Notice that the guarantees provided by Proposition 5.1 are valid irrespectively of the underlying probability distribution of the input, which may even be not known explicitly, e.g., when feeding Algorithm 1 with historical time series as realizations of the stochastic input u.

The validity of Proposition 5.1 is particularly important in domains like cloud computing. Indeed, in cloud computing it is hard to model the probability distribution of possible failures, due to the complexity of the underlying system. It is therefore more convenient to use an approach like the one presented herein that is valid independently of the underlying probability distribution.

In the next presented experiments we chose $\epsilon = 0.1$, $\eta = 0.05$, and $\beta = 10^{-7}$, corresponding to $N = 798$ scenarios (experiments) of which $\kappa = 41$ can be discarded according to Algorithm 1.

6. Empirical Evaluation

In this section we evaluate the proposed brownout-aware load-balancing algorithms through N experiments on a small-scale cloud infrastructure using the scenario theory. First, we describe our experimental setup. Next, we show the improvements that can be achieved by using our brownout-specific load-balancing algorithms.

6.1. Experimental Setup

Experiments were conducted on a single physical machine equipped with two AMD Opteron™ 6272 processors[4] and 56 GB of memory. To simulate a typical cloud environment and allow us to easily fail and restart replicas, we use the Xen hypervisor [57]. Each replica is deployed with all its tiers – web server and database server – inside its own VM, as is commonly done in practice [58], e.g., using a LAMP stack [59]. Each VM was configured with a static amount of memory, 6 GB, enough to hold all processes and the database in-memory. We deployed a total of 6 replicas, whose number of cores is 1, 1, 2, 3, 5, 8, respectively. Such a heterogeneous allocation may occur due to past service provisioning decisions, whether manual or automated [60].

Inside the replicas we deployed identical copies of RUBiS [61], an eBay-like e-commerce prototype, that is widely-used for cloud benchmarking [62, 63, 64, 65, 66, 67, 68]. RUBiS can be made brownout-compliant [9], and we added the code for piggy-backing of the dimmer value[5]. The replica controllers are equally configured, with a target 95th percentile response time of 1 second. To avoid having to deal with synchronization or consistency issues, we only used a read-only workload. However, adding consistency to replicated services is well-understood [69, 70, 71] and, in case of RUBiS, would only require an engineering effort. The load-balancer, i.e., `lighttpd` extended with our brownout-aware algorithms, was deployed inside the privileged VM in Xen, i.e., Dom0, pinned to a dedicated core.

We had three options for workload generation: open, closed or partly-open [72]. In an open system model, typically modeled as a Poisson process, requests are issued

[4] 2100 MHz, 16 cores per processor, no hyper-threading.

[5] `https://github.com/cloud-control/brownout-rubis`

with an exponentially-random inter-arrival time, characterized by a single parameter called the *arrival rate*, without waiting for requests to actually complete. In contrast, in a closed system model, a number of users access the service, each executing the following loop: issue a request, wait for the request to complete, "think" for a random time interval, repeat. The resulting average request inter-arrival time is the sum of the average think-time and the average response time of the service, hence dependent on the performance of the evaluated service. A partly-open system model is a mixture between the two: Users arrive according to a Poisson process and leave after some time, but behave closed while in the system. As with the closed model, the inter-arrival time depends on the performance of the evaluated system.

We chose to use an open system model workload generator. Since its behavior does not depend on the performance of the service, this allows us to obtain a more objective evaluation of the methods. We extended this model to include timeouts, as required to emulate users' tolerable waiting time of 4 seconds [12].

Given our chosen model and the need to measure brownout-specific behavior, the workload generator provided with RUBiS was insufficient for three reasons. First, RUBiS's workload generator uses a closed system model, without timeouts. Second, it only reports statistics for the whole experiment and does not export individual data for each request, preventing us from computing the distance as per Eq. (7). Finally, the tool cannot report whether a request has been served with or without optional content, which represents the quality of the user-experience and the revenue of the service provider. Therefore, we extended the workload generator, `httpmon`[6], to provide the needed functionalities.

We made sure that the results are reliable are little influenced by side effects as follows:

- replicas were warmed up before each experiment, i.e., all virtual disk content was cached in the VM's kernel;
- replicas were isolated performance-wise by pinning each virtual core to its own

[6] `https://github.com/cloud-control/httpmon`

physical core;

- experiments were terminated after the workload generator issued the same number of requests;
- `httpmon` and the `lighttpd` were each executed on a dedicated core;
- no non-essential processes nor `cron` scripts were running at the time of the experiments.
- `httpmon` generated the same arrival times for requests, each time it was executed, i.e., its pseudo-random number generator used to emulate the Poisson arrival process was seeded with a constant.

We generated $N = 798$ scenarios as follows. Each scenario executes for 500 seconds, which are divided in 5 equal time interval of 100 seconds. At the beginning of each time interval, one or more parameters are changed:

- With probability 50%, a new arrival rate is extracted uniform randomly in $[1, 500]$.
- With probability 50%, a uniform randomly extracted replica is either failed or restored.

As a result, the random variable constituting a scenario is composed of a tuple of 5 integer values for arrival rates, one for each time interval, and 30 boolean values for replica state (up or down), one for each replica and time interval. The execution of all 798 scenarios for three load-balancing algorithms took around two full weeks.

6.2. SQF vs. Brownout-Aware Load-Balancers

Fig. 3 shows the $\hat{\rho}^\star$ computed for the three load-balancing algorithms. According to the scenario theory, the two control-theoretical techniques perform better than SQF, and PIBH is the best solution. Indeed, they provide lower values for the distance from the ideal behavior. This means that the probability that the distance (7) between the actual behavior and the ideal one is less than the estimated $\hat{\rho}^\star$ is greater than or equal to $1 - \epsilon$.

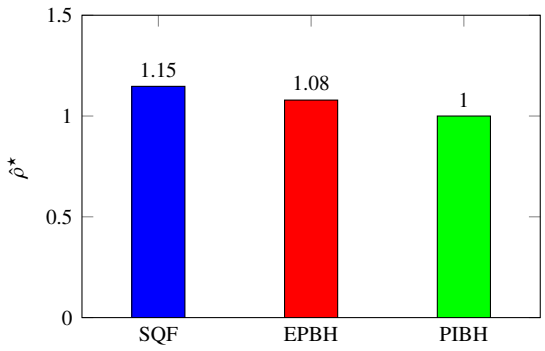

Figure 3: Results for the scenario theory.

Table 1: Statistical values for the three load-balancing algorithms.

	SQF	EPBH	PIBH
Average distance	0.5759	0.5526	**0.4892**
Std. Dev. distance	0.2696	0.2602	**0.2533**
Average % OC	44.7%	46.7%	**50.1%**
Average τ_{exc} [ms]	112.9	115.3	**81.3**

Table 2: Results of the Welch two-sample t-test.

Algorithms (OC [%])			Statistical Conclusion
SQF (44.7%)	vs	PIBH (50.1%)	PIBH significantly **better** ($p < 10^{-4}$)
SQF (44.7%)	vs	EPBH (46.7%)	EPBH significantly **better** ($p < 0.05$)

Algorithms (Avg. τ_{exc} [ms])			Statistical Conclusion
SQF (112.9 ms)	vs	PIBH (81.3 ms)	PIBH significantly **better** ($p < 10^{-2}$)
SQF (112.9 ms)	vs	EPBH (115.3 ms)	SQF **similar** to EPBH ($p = 0.4204$)

To better evaluate the performance of the three algorithms, and complementing the randomized solution of the CCP (6), Table 1 shows additional statistics. Also from this perspective PIBH exhibits the better average and the lower standard deviation of the computed distance (7). On the other hand, if we consider the average percentage of optional content, PIBH is still the algorithm with the better performance. Although PIBH is serving on average more optional content, the average τ_{exc} of its response time is better than the other two methods.

A Welch two-sample t-test, similar to the one presented in [14] confirmed that PIBH and EPBH outperform SQF in terms of optional content served, while PIBH is the only one that is significantly better than SQF with respect to τ_{exc} (see Table 2).

A further analysis on the obtained results is presented in Fig. 4, that shows, in the left plot, the boxplot of the average optional content for all the experiments; in the central plot, the boxplot of the average τ_{exc}, and in the right plot, the boxplot of the computed distance for all the experiments. This analysis breaks down the two dimensions included in the distance, showing that even in terms of optional content and of τ_{exc} separately, PIBH exhibits better performance than the other methods, while EPBH is behaving better than SQF only in terms of optional content served.

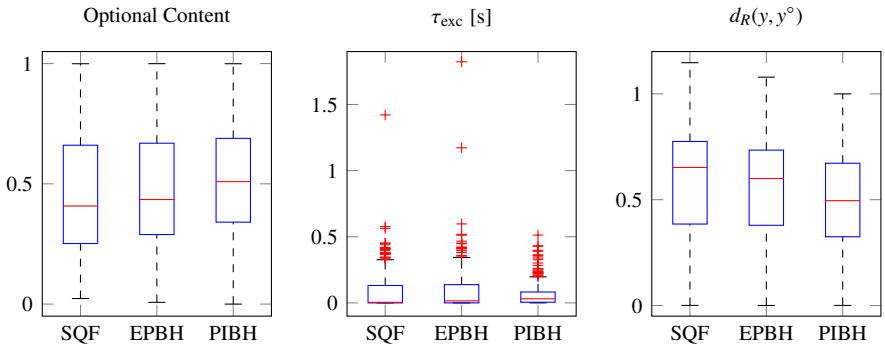

Figure 4: Boxplot for the average optional content, τ_{exc} and distance for all the performed experiments.

6.3. Discussion

The results highlight the tradeoff between the average response time and the average amount of optional content served. Indeed, EPBH is able to serve more optional content at the cost of a slight increase of the response time. However, this is not true for PIBH, which is able to serve even more optional content, while reducing τ_{exc}.

From a practical viewpoint, we could say that the improvement in terms of response time is less important than the one related to the optional content. Indeed, all the policies are able to keep τ fairly close to the setpoint, hence far from the tolerable waiting time of 4 seconds.

On the other hand, the brownout-aware load-balancing algorithms outperform the state-of-the-art by up to 5% in terms of optional content served, providing also an improvement in terms of distance (7) of about 15%. The result holds with the probabilistic guarantees provided by the scenario theory, which is something that can be used for defining proper Service Level Agreements (SLAs).

In practice, this improvement translates into better quality of experience for users and increased revenue for the service provider. Hence, our contribution helps cloud services to better cope with failure, flash crowds, and unexpected behaviors leading to capacity shortages. In other words, cloud applications become more resilient.

Noteworthy is that the competitor, SQF has been found to be near-optimal with respect to response time for non-adaptive services [21, 29]. Thus, besides improving resilience of cloud services, our contribution may be of interest to other communities,

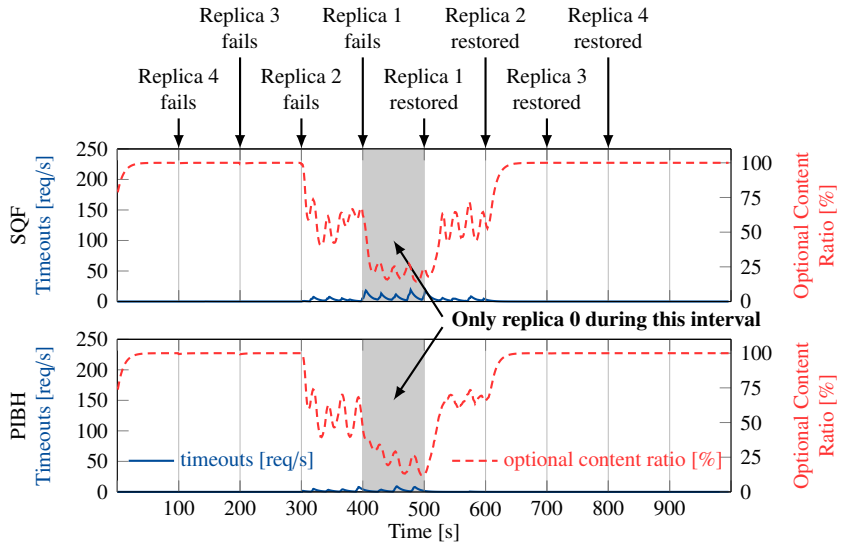

Figure 5: Experimental results comparing resilience with brownout but with different load balancers: SQF and PIBH. Configuration: 5 replicas, each having 4 cores.

to discover the limits of SQF, and sketch a possible way to design new dynamic load-balancing algorithms.

Finally, it is worth analyzing how PIBH behaves in presence of multiple failures, as we did with SQF in Section 3.3. Fig. 5 shows the same scenario used for testing SQF with and without brownout. For convenience, the top graphs reports the behavior of SQF with brownout reported in Fig. 2. The qualitative result is similar to the one obtained with SQF, but PIBH is able to serve even in this scenario 1.54% more requests with optional content, but with 65.19% less timed out requests. As a result, we can also conclude that PIBH is able to improve the reliability of the system.

7. Conclusion and Future Work

We analyzed a novel approach for improving resilience, and the ability to hide failures, in cloud services using a combination of brownout and load-balancing algorithms. The adoption of the brownout paradigm allows the service to autonomously reduce computing capacity requirements by degrading user experience in order to

guarantee that response times are bounded. Thus, it provides a natural candidate for resilience improvement when failures lead to capacity shortages. However, state-of-the-art load-balancers are generally not designed for self-adaptive cloud services. The self-adaptivity embedded in the brownout service interferes with the actions of load-balancers that route requests based on measurements of the response times of the replicas.

In order to investigate how brownout can be used for improving resilience, we extended the popular `lighttpd` web server with two new brownout-aware load-balancers. A first set of experiments showed that brownout provides substantial advantages in terms of resilience to cascading failures, even when employing SQF, a state-of-the-art, yet brownout-unaware, load-balancer. A second extensive set of experiments compared SQF to the brownout-aware load-balancers, specifically designed to act on a per-request basis. The performance evaluation was translated into a CCP which was solved by means of the scenario theory, thus providing probabilistic guarantees on the obtained result. The obtained results were also compared with a classical t-test analysis, over the large collected data-set.

The evaluation shows that with high statistical significance, the proposed solutions consistently outperform the current standards: They reduce the user experience degradation, thus perform better at hiding failures. While designed with brownout in mind, PIBH and EPBH may be useful to load-balance other self-adaptive cloud services, whose performance is not reflected in the response time or queue length.

In the future, we would like to continue with the load-balancing development, considering architectures where the load-balancer itself is replicated for better fault-tolerance. Also, we would like to react faster when events are happening at the replica level. This requires redesigning the local replica controller computing the dimmer to be event-based instead of periodic [73]. Another future research direction is the design of a holistic approach to replica control and load-balancing, extending the current replica controllers with auto-scaling features [74], that would allow the system to autonomously manage the number of replicas, together with the traffic routing, to obtain a cloud service that is both resilient and cost-effective.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.

[2] A. Gulati, G. Shanmuganathan, A. Holler, I. Ahmad, Cloud-scale resource management: challenges and techniques, in: 3rd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'11, 2011, pp. 1–5.

[3] L. A. Barroso, J. Clidaras, U. Hölzle, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2013.

[4] Q. Guan, S. Fu, Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures, in: IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013, pp. 205–214.

[5] J. Hamilton, On designing and deploying internet-scale services, in: 21st Conference on Large Installation System Administration Conference, LISA'07, 2007, pp. 18:1–18:12.

[6] M. Gallet, N. Yigitbasi, B. Javadi, D. Kondo, A. Iosup, D. H. J. Epema, in: P. D'Ambra, M. Guarracino, D. Talia (Eds.), Euro-Par 2010 – Parallel Processing, Vol. 6271 of Lecture Notes in Computer Science, Springer-Verlag, pp. 88–100.

[7] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, D. H. J. Epema, Analysis and modeling of time-correlated failures in large-scale distributed systems, in: ACM/IEEE International Conference on Grid Computing, Grid'10, ACM / IEEE Computer Society Press, 2010, pp. 355–366.

[8] E. Chuah, A. Jhumka, S. Narasimhamurthy, J. Hammond, J. Browne, B. Barth, Linking resource usage anomalies with system failures from cluster log data, in: IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013, pp. 111–120.

[9] C. Klein, M. Maggio, K.-E. Årzén, F. Hernández-Rodriguez, Brownout: Building more robust cloud applications, in: 36th International Conference on Software Engineering, ICSE 2014, ACM, 2014, pp. 700–711.

[10] M. Maggio, C. Klein, K.-E. Årzén, Control strategies for predictable brownout in cloud computing, in: 19th IFAC World Congress, 2014, pp. 689–694.

[11] D. Fleder, K. Hosanagar, A. Buja, Recommender systems and their effects on consumers: The fragmentation debate, in: 11th ACM Conference on Electronic Commerce, EC '10, ACM, 2010, pp. 229–230.

[12] F. F.-H. Nah, A study on tolerable waiting time: how long are web users willing to wait?, Behaviour and Information Technology 23 (3) (2004) 153–163.

[13] J. Dürango, M. Dellkrantz, M. Maggio, C. Klein, A. V. Papadopoulos, F. Hernández-Rodriguez, E. Elmroth, K.-E. Årzén, Control-theoretical load-balancing for cloud applications with brownout, in: IEEE 53rd Annual Conference on Decision and Control (CDC), 2014, pp. 5320–5327.

[14] C. Klein, A. V. Papadopoulos, M. Dellkrantz, J. Dürango, M. Maggio, K.-E. Årzén, F. Hernández-Rodriguez, E. Elmroth, Improving cloud service resilience using brownout-aware load-balancing, in: IEEE 33rd International Symposium on Reliable Distributed Systems (SRDS), 2014, pp. 31–40.

[15] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, A. Greenberg, Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services, Performance Evaluation 68 (11) (2011) 1056–1071, special Issue: Performance 2011.

[16] M. Lin, Z. Liu, A. Wierman, L. L. H. Andrew, Online algorithms for geographical load balancing, in: Green Computing Conference (IGCC), 2012 International, IEEE, 2012, pp. 1–10.

[17] S. Nakrani, C. Tovey, On honey bees and dynamic server allocation in internet hosting centers, Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems 12 (3-4) (2004) 223–240.

[18] A. Robertsson, B. Wittenmark, M. Kihl, M. Andersson, Design and evaluation of load control in web server systems, in: American Control Conference, Vol. 3, 2004, pp. 1980–1985.

[19] Z. Tang, J. Birdwell, J. Chiasson, C. Abdallah, M. Hayat, Resource-constrained load balancing controller for a parallel database, IEEE Trans. on Control Systems Technology 16 (4) (2008) 834–840.

[20] J. Birdwell, J. Chiasson, C. Abdallah, Z. Tang, N. Alluri, T. Wang, The effect of time delays in the stability of load balancing algorithms for parallel computations, in: 42nd IEEE Conference on Decision and Control, Vol. 1, 2003, pp. 582–587.

[21] V. Gupta, M. Harchol Balter, K. Sigman, W. Whitt, Analysis of join-the-shortest-queue routing for web server farms, Performance Evaluation 64 (9-12) (2007) 1062–1081, performance 2007 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation.

[22] G. Calafiore, M. Campi, Uncertain convex programs: randomized solutions and confidence levels, Mathematical Programming 102 (1) (2005) 25–46.

[23] M. Campi, S. Garatti, A sampling-and-discarding approach to chance-constrained optimization: Feasibility and optimality, Journal of Optimization Theory and Applications 148 (2) (2011) 257–280.

[24] A. Nemirovski, A. Shapiro, Scenario approximations of chance constraints, in: G. Calafiore, F. Dabbene (Eds.), Probabilistic and Randomized Methods for Design under Uncertainty, Springer London, 2006, pp. 3–47.

[25] G. Calafiore, M. Campi, The scenario approach to robust control design, IEEE Trans. on Automatic Control 51 (5) (2006) 742–753.

[26] M. C. Campi, S. Garatti, M. Prandini, The scenario approach for systems and control design, Annual Reviews in Control 33 (2) (2009) 149–157.

[27] A. V. Papadopoulos, M. Prandini, Model reduction of switched affine systems: A method based on balanced truncation and randomized optimization, in: 17th

International Conference on Hybrid Systems: Computation and Control, HSCC '14, ACM, 2014, pp. 113–122.

[28] K. Margellos, P. Goulart, J. Lygeros, On the road between robust optimization and the scenario approach for chance constrained optimization problems, IEEE Trans. on Automatic Control 59 (8) (2014) 2258–2263.

[29] J. Kuri, A. Kumar, Optimal control of arrivals to queues with delayed queue length information, IEEE Trans. on Automatic Control 40 (8) (1995) 1444–1450.

[30] L. Wang, V. Pai, L. Peterson, The effectiveness of request redirection on cdn robustness, SIGOPS Oper. Syst. Rev. 36 (SI) (2002) 345–360.

[31] Y. Lin, S. Kulkarni, Automated multi-graceful degradation: A case study, in: IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013, pp. 81–90.

[32] J. L. Wolf, P. S. Yu, On balancing the load in a clustered web farm, ACM Trans. Internet Technol. 1 (2) (2001) 231–261.

[33] J. A. Stankovic, An application of bayesian decision theory to decentralized control of job scheduling, IEEE Trans. on Computers C-34 (2) (1985) 117–130.

[34] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, J. Zelenka, Informed prefetching and caching, in: Fifteenth ACM Symposium on Operating Systems Principles, SOSP '95, ACM, 1995, pp. 79–95.

[35] Y. Diao, C. W. Wu, J. Hellerstein, A. Storm, M. Surenda, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, J. Colaco, Comparative studies of load balancing with control and optimization techniques, in: American Control Conference, Vol. 2, 2005, pp. 1484–1490.

[36] J. M. Bahi, S. Contassot-Vivier, R. Couturier, Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms, IEEE Trans. Parallel Distrib. Syst. 16 (4) (2005) 289–299.

[37] T. Lorido-Botrán, J. Miguel-Alonso, J. A. Lozano, A review of auto-scaling techniques for elastic applications in cloud environments, Journal of Grid Computing (2014) 1–34doi:10.1007/s10723-014-9314-7.

[38] A. Gandhi, M. Harchol-Balter, R. Raghunathan, M. A. Kozuch, Autoscale: Dynamic, robust capacity management for multi-tier data centers, ACM Trans. Comput. Syst. 30 (4) (2012) 14:1–14:26. doi:10.1145/2382553.2382556.

[39] S. Manfredi, F. Oliviero, S. Romano, A distributed control law for load balancing in content delivery networks, IEEE/ACM Trans. on Networking 21 (1) (2013) 55–68.

[40] V. Cardellini, M. Colajanni, P. S. Yu, Request redirection algorithms for distributed web systems, IEEE Trans. Parallel Distrib. Syst. 14 (4) (2003) 355–368.

[41] C. Huang, T. Abdelzaher, Bounded-latency content distribution feasibility and evaluation, IEEE Trans. on Computers 54 (11) (2005) 1422–1437.

[42] M. Andreolini, S. Casolari, M. Colajanni, Autonomic request management algorithms for geographically distributed internet-based systems, in: IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2008, pp. 171–180.

[43] S. Ranjan, R. Karrer, E. Knightly, Wide area redirection of dynamic content by internet data centers, in: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, Vol. 2, 2004, pp. 816–826.

[44] J. Doyle, R. Shorten, D. O'Mahony, Stratus: Load balancing the cloud for carbon emissions control, IEEE Trans. on Cloud Computing 1 (1) (2013) 116–128.

[45] L. Ni, K. Hwang, Optimal load balancing in a multiple processor system with many job classes, IEEE Trans. on Software Engineering SE-11 (5) (1985) 491–496.

[46] A. N. Tantawi, D. Towsley, Optimal static load balancing in distributed computer systems, J. ACM 32 (2) (1985) 445–465.

[47] M. Mitzenmacher, The power of two choices in randomized load balancing, IEEE Trans. Parallel Distrib. Syst. 12 (10) (2001) 1094–1104.

[48] T.-L. Pao, J.-B. Chen, The scalability of heterogeneous dispatcher-based web server load balancing architecture, in: Parallel and Distributed Computing, Applications and Technologies, 2006. PDCAT '06. Seventh International Conference on, IEEE Computer Society, 2006, pp. 213–216.

[49] F5, Big-ip local traffic manager (2013).
URL http://www.f5.com/products/big-ip/big-ip-local-traffic-manager/

[50] L. Zhang, Z. Zhao, Y. Shu, L. Wang, O. W. W. Yang, Load balancing of multipath source routing in ad hoc networks, in: Communications, 2002. ICC 2002. IEEE International Conference on, Vol. 5, 2002, pp. 3197–3201.

[51] H. Kameda, E.-Z. Fathy, I. Ryu, J. Li, A performance comparison of dynamic vs. static load balancing policies in a mainframe-personal computer network model, in: 39th IEEE Conference on Decision and Control, Vol. 2, IEEE, 2000, pp. 1415–1420.

[52] Y. Diao, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, Incorporating cost of control into the design of a load balancing controller, in: Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE, 2004, pp. 376–385.

[53] S. Casolari, M. Colajanni, S. Tosi, Self-adaptive techniques for the load trend evaluation of internal system resources, in: Autonomic and Autonomous Systems, 2009. ICAS '09. Fifth International Conference on, IEEE Computer Society, 2009, pp. 28–33.

[54] D. Ardagna, S. Casolari, M. Colajanni, B. Panicucci, Dual time-scale distributed capacity allocation and load redirect algorithms for clouds, Journal of Parallel and Distributed Computing 72 (6) (2012) 796–808.

[55] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's highly available key-value store, SIGOPS Oper. Syst. Rev. 41 (6) (2007) 205–220.

[56] A. Prékopa, Probabilistic programming, in: A. Ruszczyǹski, A. Shapiro (Eds.), Stochastic Programming, Vol. 10 of Handbooks in Operations Research and Management Science, Elsevier, 2003, pp. 267–351.

[57] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03, ACM, 2003, pp. 164–177.

[58] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, C. Dorai, Are clouds ready for large distributed applications?, SIGOPS Oper. Syst. Rev. 44 (2).

[59] Tutorial: Installing a LAMP web server (2013).
URL http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/install-LAMP.html

[60] M. Sedaghat, F. Hernández-Rodriguez, E. Elmroth, A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling, in: 2013 ACM Cloud and Autonomic Computing Conference, CAC '13, ACM, 2013, pp. 6:1–6:10.

[61] Rice university bidding system (2014).
URL http://rubis.ow2.org

[62] Z. Gong, X. Gu, J. Wilkes, PRESS: PRedictive Elastic ReSource Scaling for cloud systems, in: Network and Service Management (CNSM), 2010 International Conference on, IEEE, 2010, pp. 9–16.

[63] Z. Shen, S. Subbiah, X. Gu, J. Wilkes, CloudScale: elastic resource scaling for multi-tenant cloud systems, in: 2Nd ACM Symposium on Cloud Computing, SOCC '11, ACM, 2011, pp. 5:1–5:14.

[64] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, Y. Turner, JustRunIt: Experiment-based management of virtualized data centers, in: Conference on USENIX Annual Technical Conference, USENIX'09, 2009, pp. 18–28.

[65] C. Stewart, K. Shen, Performance modeling and system management for multi-component online services, in: 2nd Conference on Symposium on Networked Systems Design & Implementation - volume 2, NSDI'05, 2005, pp. 71–84.

[66] N. Vasić, D. Novaković, S. Miučin, D. Kostić, R. Bianchini, DejaVu: accelerating resource allocation in virtualized environments, in: Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XVII, ACM, 2012, pp. 423–436.

[67] C. Stewart, T. Kelly, A. Zhang, Exploiting nonstationarity for performance prediction, in: 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, EuroSys '07, ACM, 2007, pp. 31–44.

[68] Y. Chen, S. Iyer, X. Liu, D. Milojicic, A. Sahai, SLA decomposition: Translating service level objectives to system level thresholds, in: Autonomic Computing, 2007. ICAC '07. Fourth International Conference on, IEEE, 2007, pp. 1–10.

[69] N. L. Diegues, P. Romano, Bumper: Sheltering trans. from conflicts, in: IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013, pp. 185–194.

[70] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: 1st ACM Symposium on Cloud Computing, SoCC '10, ACM, 2010, pp. 143–154.

[71] M. Ardekani, P. Sutra, M. Shapiro, Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems, in: IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013, pp. 163–172.

[72] B. Schroeder, A. Wierman, M. Harchol-Balter, Open versus closed: A cautionary tale, in: 3rd Conference on Networked Systems Design & Implementation, NSDI'06, 2006, pp. 239–252.

[73] D. Desmeurs, C. Klein, A. V. Papadopoulos, J. Tordsson, Event-driven application brownout: Reconciling high utilization and low tail response times, in: 2015

IEEE International Conference on Cloud and Autonomic Computing (ICCAC), 2015, pp. 1–12.

[74] A. Ali-Eldin, J. Tordsson, E. Elmroth, An adaptive hybrid elasticity controller for cloud infrastructures, in: IEEE Network Operations and Management Symposium (NOMS), 2012, pp. 204–212.