Control-theoretical load-balancing for cloud applications with brownout

Jonas Dürango[1], Manfred Dellkrantz[1], Martina Maggio[1], Cristian Klein[2],
Alessandro Vittorio Papadopoulos[1], Francisco Hernández-Rodriguez[2], Erik Elmroth[2] and Karl-Erik Årzén[1]
[1]Department of Automatic Control, Lund University, Sweden, [2]Department of Computing Science, Umeå University, Sweden

***Abstract*— Cloud applications are often subject to unexpected events like flash crowds and hardware failures. Without a predictable behaviour, users may abandon an unresponsive application. This problem has been partially solved on two separate fronts: first, by adding a self-adaptive feature called *brownout* inside cloud applications to bound response times by modulating user experience, and, second, by introducing replicas — copies of the applications having the same functionalities — for redundancy and adding a load-balancer to direct incoming traffic.**

However, existing load-balancing strategies interfere with brownout self-adaptivity. Load-balancers are often based on response times, that are already controlled by the self-adaptive features of the application, hence they are not a good indicator of how well a replica is performing.

In this paper, we present novel load-balancing strategies, specifically designed to support brownout applications. They base their decision not on response time, but on user experience degradation. We implemented our strategies in a self-adaptive application simulator, together with some state-of-the-art solutions. Results obtained in multiple scenarios show that the proposed strategies bring significant improvements when compared to the state-of-the-art ones.

I. 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 [9], 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 [17]. Therefore, self-hosting service providers themselves are converting their computing infrastructures into small clouds.

One of the main issues with cloud computing infrastructures is **application robustness** to unexpected events. For example, flash-crowds are sudden increments of end-users, that may raise the required capacity up to five times [7]. Similarly, hardware failures may temporarily reduce the capacity of the infrastructure, while the failure is repaired [5]. Also, unexpected performance degradations may arise due to workload consolidation and the resulting interference among co-located applications [27]. Due to the large magnitude and short duration of such events, it may be economically too costly to keep enough spare capacity to properly deal with them. As a result, unexpected events may lead to infrastructure overload, that translates to unresponsive services, leading to dissatisfied end-users and revenue loss.

Cloud services therefore greatly benefit from self-adaptation techniques [35], such as **brownout** [21, 25]. A brownout service adapts itself by reducing the amount of computations it executes to serve a request, so as to maintain response time around a given setpoint. In essence, some computations are marked as mandatory — for example, displaying product information in an e-commerce website — while others are optional — for example, recommending similar products. Whenever an end-user request is received, the service can choose to execute the optional code or not according to its available capacity, and to the previously measured response times. Note that executing optional code directly translates into a better service for the end-user and more revenue for the service provider. This approach has proved to be successful for dealing with unexpected events [21]. However, there, brownout services were composed of a single *replica*, i.e., a single copy of the application, running inside a single VM.

In this paper, we extend the brownout paradigm to services featuring multiple replicas — i.e., multiple, independent copies of the same application, serving the user the same data — hosted inside individual VMs. Since each VM can be hosted by different PMs, this enhances brownout services in two directions. First, *scalability* of a brownout application — the ability for an application to deal with more users by adding more computing resources — is improved, since applications are no longer limited to using the resources of a single PM. Second, resilience is improved: in case a PM fails, taking down a replica, other replicas whose VMs are hosted on different PMs can seamlessly take over.

The component that decides which replica should serve a particular end-user request is called a *load-balancer*. Despite the fact that load-balancing techniques have been widely studied [5, 23, 24, 29], 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. Therefore, the novelty of our problem consists in finding a brownout-compliant load-balancing technique that is aware of each replica's self-adaptation mechanism.

The contribution of this paper is summarized as follows.

- We present extensions to load-balancing architectures and the required enhancements to the replicas that convey information about served optional content and allow

Corresponding author: jonas.durango@control.lth.se. 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.

to deal with brownout services efficiently (Section III).
- We propose novel load-balancing algorithms that, by receiving information about the adaptation happening at the replica level, try to maximize the performance of brownout services, in terms of frequency of execution of the optional code (Section IV).
- We show through simulations that our brownout-aware load-balancing algorithms outperform state-of-the-art techniques (Section V).

II. Related Work

Load-balancers are standard components of Internet-scale services [40], allowing applications to achieve scalability and resilience [5, 18, 41]. Many load-balancing policies have been proposed, aiming at different optimizations, spanning from equalizing processor load [37] to managing memory pools [13, 32], to specific optimizations for iterative algorithms [4]. Often load-balancing policies consider web server systems as a target [11, 26], where one of the most important result is to bound the maximum response time that the clients are exposed to [19]. Load-balancing strategies can be guided by many different purposes, for example geographical [2, 33], driven by the electricity price to reduce the datacenter operation cost [15], or specifically designed for cloud applications [5, 23, 24].

Load-balancing solutions can be divided into two different types: static and dynamic. Static load-balancing refers to a fixed, non-adaptive strategy to select a replica to direct traffic to [30, 38]. 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 a replica at **Random**. However, due to their static nature, such techniques would not have good performance when applied to brownout-compliant applications as they do not take into account the inherent fluctuations of a cloud environment and the control strategy at the replica level, which leads to changing capabilities of replicas.

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)** [28], that randomly chooses two replicas and assigns the request to the fastest one, i.e., the one with the lowest maximum response time.

Through experimental results, we were able to determine that FRF, FRF-EWMA and 2RC are unsuitable for brownout applications. They base their decision on response times alone, which leads to inefficient decisions for brownout services. Indeed, such services already keep their response-time at a given setpoint, at the expense of reducing the ratio of optional content served. Hence, by measuring response-time alone, it is not possible to discriminate between a replica that is avoiding overload by not executing the optional code and a replica that is not subject to overload executing all optional code, both achieving the desired response times.

Another adopted strategy is based on the pending request count and generally called **Shortest Queue First (SQF)**, where the load-balancer tracks the pending requests and select the replicas with the least number of requests waiting for completion. This strategy pays off 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 [31]. 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 [6], 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 as follows: A **Predictive** load balancer would rank the replicas based on the difference between the past metrics and the current ones. 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 our solution observes the changes in the ratio of optional code served and tries to maximize the requests served enabling the full computation.

Dynamic solutions can be control-theoretical [20, 42] and also account for the cost of applying the control action [14] or for the load trend [12]. 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 execute, like in [3]. 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 [21].

III. Problem Statement

Load-balancing problems can be formulated in many ways. This is especially true for the case addressed in this paper where the load-balancer should distribute the load to adaptive entities, that play a role by themselves in adjusting to the current situation. This section discusses the characteristics of the considered infrastructure and clearly formulates the problem under analysis.

Figure 1 illustrates the software architecture that is deployed to execute a brownout-compliant application composed of multiple replicas. Despite the modifications needed

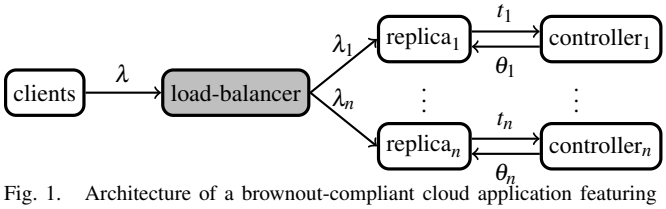

Fig. 1. Architecture of a brownout-compliant cloud application featuring multiple replicas.

to make it brownout-compliant, the architecture is widely accepted as the reference one for cloud applications [5].

Given the generic cloud application architecture, access can only be done through the load-balancer. The clients are assumed to be closed-loop: They first send a request, wait for the reply, then think by waiting for an exponentially distributed time interval, and repeat. This client model is a fairly good approximation for users that interact with websites requiring a pre-defined number of requests to complete a goal, such as buying a product [16] or booking a flight. The resulting traffic has an unknown but measurable rate λ.

Each client request is received by the load-balancer, that sends it to one of the n replicas. The chosen replica produces the response and sends it back to the load-balancer, which forwards it to the original client. We measure the **response time** of the request as the time spent within the replica, assuming negligible time is taken for the load-balancer execution and for the routing itself. Since the responses are routed back to the load-balancer, it is possible to attach information to be routed back to aid balancing decisions to it.

Each replica i receives a fraction λ_i of the incoming traffic and is a stand-alone version of the application. More specifically, each replica receives requests at a rate $\lambda_i = w_i \cdot \lambda$, such that $w_i \geq 0$, and $\sum_i w_i = 1$. In this case, the load balancer simply computes the **replica weights** w_i according to its load-balancing policy.

Special to our case is the presence of a controller within each replica [21]. This controller takes care of adjusting the percentage of requests θ_i served with the optional components enabled, based on the measured response time t_i of the requests served by the replica. The controller for replica i receives statistics on the replica response times. The average, 95-th percentile, or maximum value can be used, depending on the requirements of the application. Here we use the 95-th percentile of response times.

As given by the brownout paradigm, a replica i responds to requests either partially, where only mandatory content is included in the reply, or fully, where both mandatory and optional content is included. This decision is taken independently for each request, based on a Bernoulli trial, with probability θ_i for success. The service rate for a partial response is μ_i while a full response is generated with a rate M_i. Obviously, partial replies are faster to compute than full ones, since the optional content does not need to be prepared, hence, $\mu_i \geq M_i$. Assuming the replica is not saturated, it serves requests fully at a rate $\lambda_i \theta_i$ and partially at a rate $\lambda_i(1-\theta_i)$.

Many alternatives can be envisioned on how to extend existing load balancers to deal with brownout-compliant applications. In our choice, the load-balancer receives information about θ_i from the replicas. This solution results in less computationally intensive load-balancers with respect to the case where the load-balancer should somehow estimate the probability of executing the optional components, but requires additional communication. The overhead, however, is very limited, since only one value would be reported per replica. For the purpose of this paper, we assume that to aid load-balancing decisions, each replica piggy-backs the current value of θ_i through the reply, so that this value can be observed by the load-balancer, limiting the overhead. The load-balancer does not have any knowledge on *how* each replica controller adjusts the percentage θ_i, it only knows the reported value.

This allows to completely separate the action of the load-balancer from the one of the self-adaptive application. The replica control period is set to 0.5s while the load-balancer is supposed to act every second. The replica controller's pole is set to 0.99.

Given this last architecture, we want to solve the problem of designing a **load-balancer policy**. Knowing the values of θ_i for each replica $i \in [1,n]$, a load-balancer should compute the values of the weights w_i such that

$$\sum_{k=0}^{\infty} \sum_{i} w_i(k)\theta_i(k) \tag{1}$$

is maximized, where k denotes the discrete time. Given that we have no knowledge of the evolution in time of the involved quantities, we aim to maximize the quantity $\sum_i w_i \theta_i$ in every time instant, assuming that this will maximize the quantity defined in Equation (1). In other words, the load-balancer should maximize the ratio of requests served with the optional part enabled. For that, the aim is to maximize the ratio of optional components served in any time instant. In practice, this would also maximize the application owner's revenue [21].

IV. Solution

This section describes three different solutions for balancing the load directed to self-adaptive brownout-compliant applications composed of multiple replicas. The first two strategies are heuristic solutions that take into account the self-adaptivity of the replicas. The third alternative is based on optimization, with the aim of providing guarantees on the best possible behavior.

A. Variational principle-based heuristic (VPBH)

Our first solution is inspired by the predictive approach described in Section II. The core of the predictive solution is to examine the variation of the involved quantities. While in its classical form, this solution relies on variations of response times or pending request count per replica, our solution is based on how the control variables θ_i are changing.

If the percentage θ_i of optional content served is increasing, the replica is assumed to be less loaded, and more traffic can be sent to it. On the contrary, when the optional content

decreases, the replica will receive less traffic, to decrease its load and allow it to increase θ_i.

The replica weights w_i are initialized to $1/n$ where n is the number of replicas. The load-balancer periodically updates the values of the weights based on the values of θ_i received by the replicas. At time k, denoting with $\Delta\theta_i(k)$ the variation $\theta_i(k) - \theta_i(k-1)$, the solution computes a potential weight $\tilde{w}_i(k+1)$ according to

$$\tilde{w}_i(k+1) = w_i(k) \cdot [1 + \gamma_P \Delta\theta_i(k) + \gamma_I \theta_i(k)], \qquad (2)$$

where γ_P and γ_I are constant gains, respectively related to a proportional and an integral load-balancing action. As calculated, $\tilde{w}_i$ values can be negative. This is clearly not feasible, therefore negative values are truncated to a small but still positive weight ε. Using a positive weight instead of zero allows us to probe the replica and see whether it is favorably responding to new incoming requests or not. Moreover, the computed values do not respect the constraint that their sum is equal to 1, so they are then re-scaled according to

$$w_i(k) = \frac{\max(\tilde{w}_i(k), \varepsilon)}{\sum_i \max(\tilde{w}_i(k), \varepsilon)}. \qquad (3)$$

We selected $\gamma_P = 0.5$ based on experimental results. Once γ_P is 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 signal, 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 chose $\gamma_I = 5.0$.

B. Equality principle-based heuristic (EPBH)

The second policy is based on the heuristic that a near-optimal situation is when all replica serves the same percentage optional content. Based on this assumption, the control variables θ_i should be as close as possible to one another. If the values of θ_i converge to a single value, this means that the traffic is routed so that each replica can serve the same percentage of optional content, i.e., a more powerful replica receives more traffic then a less powerful one. This approach therefore selects weights that encourages the control variables θ_i to converge towards the mean $\frac{1}{n}\sum_j \theta_j$.

The policy computes a potential weight $\tilde{w}_i(k+1)$

$$\tilde{w}_i(k+1) = w_i(k) + \gamma_e \left(\theta_i(k) - \frac{1}{n} \sum_j \theta_j(k) \right) \qquad (4)$$

where γ_e is a strictly positive parameter which accounts for how fast the algorithm should converge. For the experiments we chose $\gamma_e = 0.025$. The weights are simply modified proportionally to the difference between the current control value and the average control value set by the replicas. Clearly, the same saturation and normalization described in Equation (3) has to be applied to the proposed solution, to ensure that the sum of the weights is equal to one and that they have positive values — i.e., that all the incoming traffic is directed to the replicas and that each replica receives at least some requests.

C. Convex optimization based load-balancing (COBLB)

The third approach is to update the replica weights based on the solution of an optimization problem, where the objective is to maximize the quantity $\sum_i w_i \theta_i$.

In this solution, each replica is modeled as a queuing system using a Processor Sharing (PS) discipline. The clients are assumed to arrive according to a Poisson process with intensity λ_i, and will upon arrival enter the queue where they will receive a share of the replicas processing capability. The simplest queueing models assume the required time for serving a request to be exponentially distributed with rate $\tilde{\mu}$. However, in the case of brownout, the requests are served either with or without optional content with rates M_i and μ_i, respectively. Therefore the distribution of service times S_i for the replicas can be modelled as a mixture of two exponential distributions with a probability density function $f_{S_i}(t)$ according to

$$f_{S_i}(t) = (1 - \theta_i) \cdot \mu_i \cdot e^{-\mu_i \cdot t} + \theta_i \cdot M_i \cdot e^{-M_i \cdot t}, \qquad (5)$$

where t represents the continuous time and θ_i is the probability of activating the optional components. Thus, a request entering the queue of replica i will receive an exponentially distributed service time with a rate with probability θ_i being M_i, and probability $1 - \theta_i$ being μ_i. The resulting queueing system model is of type $M/G/1/PS$ and has been proven suitable to simulate the behavior of web servers [10].

It is known that for $M/G/1$ queueing systems adopting the PS discipline, the mean response times will depend on the service time distribution only through its mean [22, 34], here given for each replica by

$$\mu_i^* = \frac{1}{\mathbb{E}[S_i]} = \left[\frac{1 - \theta_i}{\mu_i} + \frac{\theta_i}{M_i} \right]^{-1}. \qquad (6)$$

The mean response times for a $M/G/1/PS$ system themselves are given by

$$\tau_i = \frac{1}{\mu_i^* - \lambda w_i}. \qquad (7)$$

The required service rates μ_i^* needed to ensure that there is no stationary error can be obtained by inverting Equation (7)

$$\mu_i^* = \frac{1 + \tau_i^* \lambda w_i}{\tau_i^*} \qquad (8)$$

with τ_i^* being the set point for the response time of replica i.

Combining Equation (6) and (8), it is then possible to calculate the steady-state control variables θ_i^* that gives the desired behavior

$$\theta_i^* = \frac{M_i \cdot (\mu_i \tau_i^* - 1 - \lambda w_i \tau_i^*)}{(1 + \lambda w_i \tau_i^*) \cdot (\mu_i - M_i)} = \frac{A_i - B_i w_i}{C_i + D_i w_i}. \qquad (9)$$

with A_i, B_i, C_i and D_i all positive. Note that the values of θ_i^* are not used in the replicas and are simply computed by the

optimization based load-balancer as the optimal stationary conditions for the control variables θ_i. Clearly, one could also think of using these values within the replicas but in this investigation we want to completely separate the load-balancing policy and the replicas internal control loops.

Recalling that θ_i is the probability of executing the optional components when producing the response, the values θ_i^* should be constrained to belong to the interval $[0,1]$, yielding the following inequalities (under the reasonable assumptions that $\tau_i^* > 1/M_i$ and $\mu_i \geq M_i$)

$$\frac{A_i - C_i}{B_i + D_i} \leq w_i \leq \frac{A_i}{B_i}. \tag{10}$$

Using these inequalities as constraints, it is possible to formally state the optimization problem as

$$\begin{aligned} \max_{w_i} \quad & J = \textstyle\sum_i w_i \theta_i = \sum_i w_i \dfrac{A_i - B_i w_i}{C_i + D_i w_i} \\ \text{s.t.} \quad & \textstyle\sum_i w_i = 1, \\ & \dfrac{A_i - C_i}{B_i + D_i} \leq w_i \leq \dfrac{A_i}{B_i}. \end{aligned} \tag{11}$$

Since the objective function J is concave and the constraints linear in w_i, the entire problem is concave and can be solved using efficient methods [8]. We use an interior point algorithm, implemented in CVXOPT[1], a Python library for convex optimization problems, to obtain the values of the weights.

Notice that solving optimization problem (11) guarantees that the best possible solution is found for the single time instant problem, but requires a lot of knowledge about the single replicas. In fact, while other solutions require knowledge only about the incoming traffic and the control variables for each replica, the optimization-based solution relies on knowledge of the service time of requests with and without optional content M_i and μ_i that might not be available and could require additional computations to be estimated correctly.

V. Evaluation

In this section we describe our experimental evaluation, discussing the performance indicators used to compare different strategies, the simulator developed and used to emulate the behavior of brownout-compliant replicas driven by the load-balancer, and our case studies.

A. Performance indicators

Performance measures are necessary to objectively compare different algorithms. Our first performance indicator is defined as the **percentage** $\%_{oc}$ of the total requests served with the optional content enabled, which is a reasonable metric given that we assume that users perform a certain number of clicks to use the application.

We also would like to introduce some other performance metrics to compare the implemented load-balancing techniques. For this, we use the **user-perceived stability** σ_u [2]. This metric refers to the variation of performance as observed by the users, and it is measured as the standard deviation of response times. Its purpose is to measure the ability of the replicas to respond timely to the client requests. The entire brownout framework aims at stabilizing the response times, therefore it should achieve better user-perceived stability, regardless of the presence of the load-balancer. However, the load-balancing algorithm clearly influences the perceived response times, therefore it is logical to check whether the newly developed algorithms achieve a better perceived stability than the classical ones. Together with the value of the user-perceived stability, we also report the **average response time** μ_u to distinguish between algorithms that achieve a low response time with possibly high fluctuations from solutions that achieve a higher but more stable response time.

B. Simulator

To test the load-balancing strategies, a Python-based simulator for brownout-compliant applications is used. In the simulator, it is easy to plug-in new load-balancing algorithms. The simulator is based on the concepts of *Client*, *Request*, *LoadBalancer* and *Replica*.

When a new client is defined, it can behave according to the open-loop client model, where it simply issues a certain number of unrelated requests (as it is true for clients that respect the Markovian assumption), or according to the closed-loop one [1, 36]. Closed-loop clients issue a request and wait for the response, when they receive the response they think for some time (in the simulations this time is exponentially distributed with mean 1s) and subsequently continue sending another request to the application. While this second model is more realistic, the first one is still useful to simulate the behavior of a large number of clients. The simulator implements both models, to allow for complete tests, but we will evaluate our results with closed-loop clients given the nature of the applications, that requires users to perform a certain number of clicks.

Requests are received by the load-balancer, that directs them towards different replicas. The load-balancer can work on a per-request basis or based on weights. The first case is used to simulate policies like Round Robin, Random, Shortest Queue First and so on, that do not rely on the concept of weights. The weighted load-balancer is used to simulate the strategies proposed in this paper.

Each replica simulates the computation necessary to serve the request and chooses if it should be executed with or without the optional components activated. If the optional content is served the service time is a random number from a gaussian distribution with mean ϕ_i and variance 0.01, while if the optional content is not served, the mean is ψ_i and the variance is 0.001. The parameters ϕ_i and ψ_i are specified when replicas are created and can be changed during the execution. The service rate of requests with the optional component is $M_i = 1/\phi_i$ while for serving only the mandatory part of the request the service rate is $\mu_i = 1/\psi_i$. The replicas are also executing an internal control loop to select their

[1] http://cvxopt.org/

control variables θ_i [21]. The replicas use PS to process the requests in the queue, meaning that each of the n active requests will get $1/n$ of the processing capability of the replica.

The simulator receives as input a *Scenario*, which describes what can happen during the simulation. The scenario definition supports the insertion of new clients and the removal of existing ones. It also allows to turn on and off replicas at specific times during the execution and to change the service times for every replica, both for the optional components and for the mandatory ones. This simulates a change in the amount of resources given to the machine hosting the replica and it is based on the assumption that these changes are unpredictable and can happen at the architecture level, for example due to the cloud provider co-locating more applications onto the same physical hardware, therefore reducing their computation capability [39].

With the scenarios, it is easy to simulate different working conditions and to have a complete overview of the changes that might happen during the load-balancing and replica execution. In the following, we describe two experiments conducted to compare the load-balancing strategies when subject to different execution conditions.

C. Reacting to client behavior

The aim of the first test is to evaluate the performance of different algorithms when new clients arrive and existing clients disconnect.

In the experiment the infrastructure is composed of four replicas. The first replica is the fastest one and has $\phi_1 = 0.05$s (average time to execute both the mandatory and the optional components) and $\psi_1 = 0.005$s (average time to compute only the mandatory part of the response). The second replica is slower, with $\phi_2 = 0.25$s and $\psi_2 = 0.025$s. The third and fourth replicas are the slowest ones, having $\phi_{3,4} = 0.5$s and $\psi_{3,4} = 0.05$s.

Clients adhere to the closed-loop model. 50 clients are accessing the system at time 0s, and 10 of them are removed after 200s. At time 400s, 25 more clients query the application and 25 more arrives again at 600s. 40 clients disconnect at time 800s and the simulation is ended at time 1000s.

The right column in Figure 2 shows the control variable θ_i for each replica, while the left column shows the effective weights w_i, i.e., the weights that have been assigned by the load-balancing strategies computed a posteriori. Since solutions like RR do not assign directly the weights, we decided to compute the effective values that can be found after the load-balancing assignments.

The algorithms are ordered by decreasing percentage $\%_{oc}$ of optional content served, where EPBH achieves the best percentage overall, followed by VPBH and by COBLB.

For this scenario, the strategies that are aware of the adaptation at the replica level achieve better results in terms of percentage of optional content served. The SQF algorithm is the only existing one capable of achieving similar (yet lower) performance in terms of optional content delivered.

To analyze the effect of the load-balancing strategies on

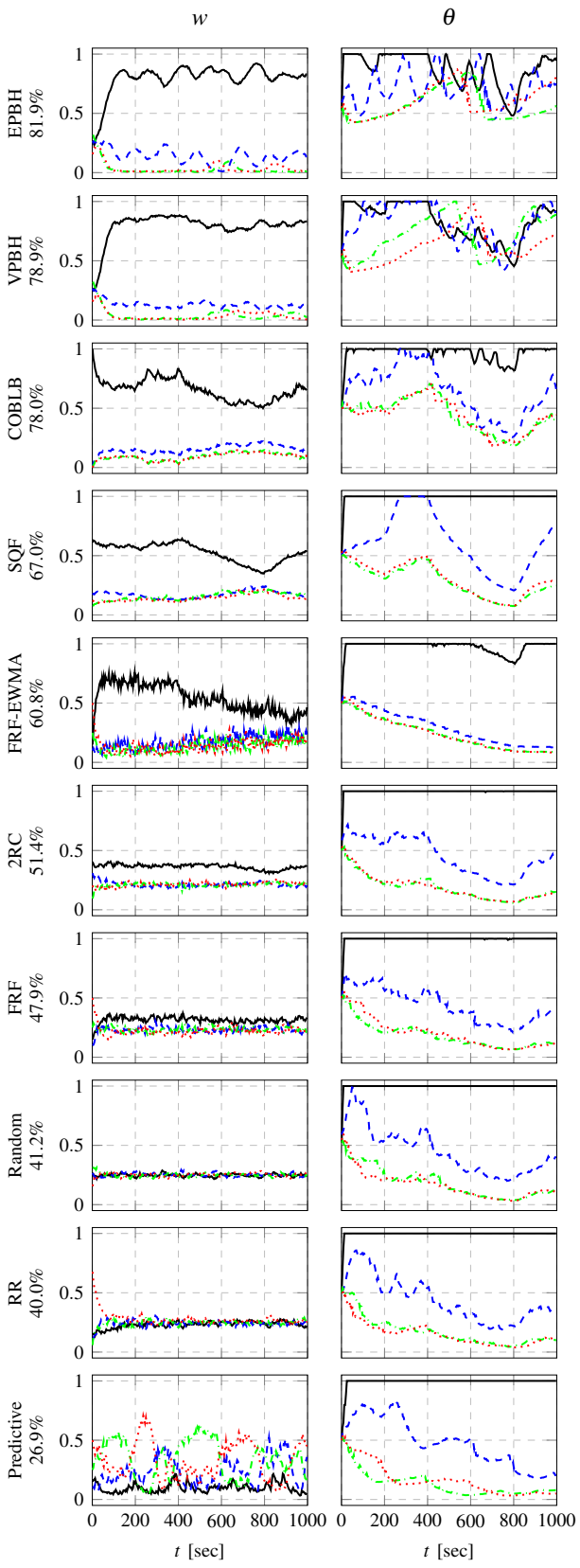

Fig. 2. Results of a simulation with four replicas and clients entering and leaving the system at different time instants. The left column shows the effective weights while the right column shows the control variables for each replica. The first replica is shown in black solid lines, the second in blue dashed lines, the third in green dash-dotted lines, and the fourth in red dotted lines.

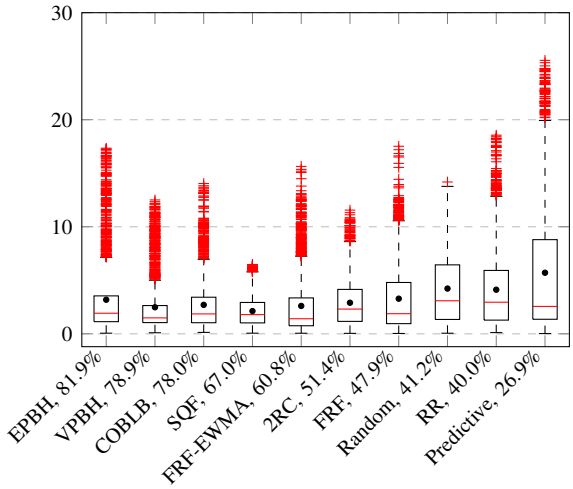

Fig. 3. Box plots of the maximum response time in all the replicas for every control interval. Each box shows from the first quartile to the third. The red line shows the median; outliers are represented with red crosses while the black dots indicate the average value (also considering the outliers).

the replicas response times, Figure 3 shows box plots of the maximum response time experienced by the replicas. The load-balancing strategies are ordered from left to right based on the percentage of optional code $\%_{oc}$ achieved. The bottom line of each box represents the first quartile, the top line the third and the red line is the median. The red crosses show the outliers. In addition to the classical box plot information, the black dots show for each algorithm the average value of the maximum response time measured during the experiment, also considering the outliers.

The box plots clearly show that all the solutions presented in this paper achieve distributions that have outliers, as well as almost all the literature ones. The only exception seems to be SQF, that achieves very few outliers, predictable maximum response time, with a median that is just slightly higher than the one achieved by VPBH. EPBH offers the highest percentage of optional content served, by sacrificing the response time bound. From this additional information one can conclude that the solutions presented in this paper should be tuned carefully if response time requirements are hard. For example, for certain tasks, users prefer a very responsive applications instead of many features, hence the revenue of the application owner may be increased through lower response times. Notice that the proposed heuristics (EPBH and VPBH) have tunable parameters that can be used to exploit the trade-off between response time bounds and optional content.

This case study features only a limited number of replicas. However, we have conducted additional tests, also in more complex scenarios, featuring up to 20 replicas, reporting results similar to the ones presented herein. In the next section we test the effect of infrastructural changes to load-balancing solutions and response times.

D. Reacting to infrastructure resources

In the second case study the architecture is composed of five replicas. At time 0s, the first replica has $\phi_1 = 0.07$s, $\psi_1 = 0.001$s. The second and third replicas are medium fast, with $\phi_{2,3} = 0.14$s and $\psi_{2,3} = 0.002$s. The fourth and fifth replicas are the slowest with $\phi_{4,5} = 0.7$s and $\psi_{4,5} = 0.01$s.

At time 250s the amount of resources assigned to the first replica is decreased, therefore $\phi_1 = 0.35$s and $\psi_1 = 0.005$s. At time 500s, the fifth replica receives more resources, achieving $\phi_5 = 0.07$s and $\psi_5 = 0.001$s. The same happens at time 750 to the fourth replica.

TABLE I
PERFORMANCE WITH VARIABLE INFRASTRUCTURE RESOURCES

Algorithm	$\%_{oc}$	μ_u	σ_u
COBLB	**90.9%**	0.78	0.97
EPBH	89.5%	1.06	1.95
VPBH	87.7%	1.02	1.90
SQF	83.3%	**0.55**	**0.40**
RR	75,5%	1.11	2.42
Random	72.9%	0.86	2.23
2RC	72.2%	0.74	1.64
FRF	70.4%	1.27	2.03
FRF-EWMA	51.4%	1.44	3.41
Predictive	47.4%	1.66	3.48

Table I reports the percentage $\%_{oc}$, the average response time and the user-perceived stability for the different algorithms. It should be noted again that our strategies obtain better optional content served at the expense of slightly higher response times. However, COBLB is capable of obtaining both low response times and high percentage of optional content served. This is due to the amount of information that it uses, since we assume that the computation times for mandatory and optional part are known. The optimization-based strategy is capable of reacting fast to changes and achieves predictability in the application behavior. Again, if one does not have all the necessary information available, it is possible to implement strategies that would better exploit the trade-off between bounded response time and optional content.

VI. CONCLUSION

We have revisited the problem of load-balancing different replicas in the presence of self-adaptivity inside the application. This is motivated by the need of cloud applications to withstand unexpected events like flash crowds, resource variations or hardware changes. To fully address these issues, load-balancing solutions need to be combined with self-adaptive applications, such as brownout. However, simply combining them without special support leads to poor performance.

Three load-balancing strategies are described, specifically designed to support brownout-compliant cloud applications. The experimental results clearly show that incorporating the application adaptation in the design of load balancing strategies pay off in terms of predictable behavior and maximized performance. They also demonstrated that the SQF algorithm is the best non-brownout-aware solution and therefore it should be used whenever it is not possible to adopt one of our proposed solution. The granularity of the actuation of the SQF load-balancing strategy is on a per-request based and the used information are much more updated with respect to the current infrastructure status, which is an advantage compared

to weight-based solutions and helps SQF to serve requests faster. In future work we plan to investigate brownout-aware per-request solutions.

Finally, the application model used in this paper assumes a finite number of clicks per user, therefore the developed load-balancer strategies maximize the percentage of optional content served. However, when a different application model is taken into account, optimizing the absolute number of requests served with optional content is another possible goal, that should be investigated in future work.

REFERENCES

[1] F. Alomari and D. Menascé. "Efficient Response Time Approximations for Multiclass Fork and Join Queues in Open and Closed Queuing Networks". In: *Parallel and Distributed Systems, IEEE Transactions on* 99 (2013), pp. 1–6.

[2] M. Andreolini, S. Casolari, and M. Colajanni. "Autonomic Request Management Algorithms for Geographically Distributed Internet-Based Systems". In: *SASO*. 2008.

[3] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci. "Dual Time-scale Distributed Capacity Allocation and Load Redirect Algorithms for Clouds". In: *J. Parallel Distrib. Comput.* 72.6 (2012).

[4] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. "Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms". In: *IEEE Trans. Parallel Distrib. Syst.* 16.4 (Apr. 2005).

[5] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool, 2009.

[6] *BIG-IP Local Traffic Manager*. `http://www.f5.com/products/big-ip/big-ip-local-traffic-manager/`. Accessed: 2013-12-31.

[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: *SOCC*. 2010.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787.

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility". In: *Future Generation Computer Systems* 25.6 (2009).

[10] J. Cao, M. Andersson, C. Nyberg, and M. Kihl. "Web server performance modeling using an M/G/1/K*PS queue". In: *10th International Conference on Telecommunications ICT 2003*. Vol. 2. 2003, pp. 1501–1506.

[11] V. Cardellini, M. Colajanni, and P. S. Yu. "Request Redirection Algorithms for Distributed Web Systems". In: *IEEE Trans. Parallel Distrib. Syst.* 14.4 (Apr. 2003).

[12] S. Casolari, M. Colajanni, and S. Tosi. "Self-Adaptive Techniques for the Load Trend Evaluation of Internal System Resources". In: *ICAS*. 2009.

[13] Y. Diao, C. W. Wu, J. Hellerstein, A. Storm, M. Surenda, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco. "Comparative studies of load balancing with control and optimization techniques". In: *ACC*. 2005.

[14] Y. Diao, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. "Incorporating cost of control into the design of a load balancing controller". In: *RTAS*. 2004.

[15] J. Doyle, R. Shorten, and D. O'Mahony. "Stratus: Load Balancing the Cloud for Carbon Emissions Control". In: *Cloud Computing, IEEE Transactions on* 1.1 (2013).

[16] D. F. García and J. García. "TPC-W E-Commerce Benchmark Evaluation". In: *Computer* 36.2 (Feb. 2003), pp. 42–48.

[17] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad. "Cloud-scale resource management: challenges and techniques". In: *HotCloud*. 2011.

[18] J. Hamilton. "On designing and deploying internet-scale services". In: *LISA*. 2007.

[19] C. Huang and T. Abdelzaher. "Bounded-latency content distribution feasibility and evaluation". In: *IEEE Transactions on Computers* 54.11 (2005).

[20] H. Kameda, E.-Z. Fathy, I. Ryu, and J. Li. "A performance comparison of dynamic vs. static load balancing policies in a mainframe-personal computer network model". In: *CDC*. 2000.

[21] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez. "Brownout: Building more Robust Cloud Applications". In: *ICSE*. (Accepted) Available as preprint at: `http://goo.gl/0jMz9S`. 2014.

[22] L. Kleinrock. "Time–shared systems: A theoretical treatment". In: *Journal of the ACM* 14.242-261 (1967).

[23] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew. "Online algorithms for geographical load balancing". In: *IGCC*. 2012.

[24] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg. "Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services". In: *Perform. Eval.* 68.11 (Nov. 2011).

[25] M. Maggio, C. Klein, and K.-E. Årzén. "Control strategies for predictable brownout in Cloud Computing". In: *IFAC WC*. (Accepted) Available as preprint at: `http://goo.gl/S9rv3d`. 2014.

[26] S. Manfredi, F. Oliviero, and S. Romano. "A Distributed Control Law for Load Balancing in Content Delivery Networks". In: *IEEE/ACM Transactions on Networking* 21.1 (2013).

[27] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. "Bubble-Up: increasing utilization in modern warehouse scale computers via sensible co-locations". In: *MICRO*. 2011, pp. 248–259.

[28] M. Mitzenmacher. "The Power of Two Choices in Randomized Load Balancing". In: *IEEE Trans. Parallel Distrib. Syst.* 12.10 (Oct. 2001).

[29] S. Nakrani and C. Tovey. "On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers". In: *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems* 12.3-4 (Sept. 2004).

[30] L. Ni and K. Hwang. "Optimal Load Balancing in a Multiple Processor System with Many Job Classes". In: *IEEE Transactions on Software Engineering* 11.5 (1985).

[31] T.-L. Pao and J.-B. Chen. "The Scalability of Heterogeneous Dispatcher-Based Web Server Load Balancing Architecture". In: *PDCAT*. 2006.

[32] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. "Informed Prefetching and Caching". In: *SOSP*. 1995.

[33] S. Ranjan, R. Karrer, and E. Knightly. "Wide area redirection of dynamic content by Internet data centers". In: *INFOCOM*. 2004.

[34] M. Sakata, S. Noguchi, and J. Oizumi. "An Analysis of the M/G/1 Queue under Round-Robin Scheduling". In: *Operations Research* 19.2 (1971), pp. 371–385.

[35] M. Salehie and L. Tahvildari. "Self-adaptive Software: Landscape and Research Challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2 (May 2009).

[36] B. Schroeder, A. Wierman, and M. Harchol-Balter. "Open Versus Closed: A Cautionary Tale". In: *NSDI*. 2006.

[37] J. A. Stankovic. "An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling". In: *IEEE Trans. Comput.* 34.2 (Feb. 1985).

[38] A. N. Tantawi and D. Towsley. "Optimal Static Load Balancing in Distributed Computer Systems". In: *J. ACM* 32.2 (Apr. 1985).

[39] L. Tomás and J. Tordsson. "Improving Cloud Infrastructure Utilization Through Overbooking". In: *CAC*. 2013, pp. 1–10.

[40] L. Wang, V. Pai, and L. Peterson. "The Effectiveness of Request Redirection on CDN Robustness". In: *OSDI*. 2002.

[41] J. L. Wolf and P. S. Yu. "On Balancing the Load in a Clustered Web Farm". In: *ACM Trans. Internet Technol.* 1.2 (Nov. 2001).

[42] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. W. W. Yang. "Load balancing of multipath source routing in ad hoc networks". In: *ICC*. 2002.