2017 IEEE 56th Annual Conference on Decision and Control (CDC)
December 12-15, 2017, Melbourne, Australia

Power-Aware Cloud Brownout: response time and power consumption control

Alessandro Vittorio Papadopoulos[1], Jakub Krzywda[2], Erik Elmroth[2], Martina Maggio[3]
[1] *IDT, Mälardalen University, Sweden*
[2] *Department of Computing Science, Umeå University, Sweden*
[3] *Department of Automatic Control, Lund University, Sweden*

***Abstract*— Cloud computing infrastructures are powering most of the web hosting services that we use at all times. A recent failure in the Amazon cloud infrastructure made many of the website that we use on a hourly basis unavailable[1]. This illustrates the importance of cloud applications being able to absorb peaks in workload, and at the same time to tune their power requirements to the power and energy capacity offered by the data center infrastructure. In this paper we combine an established technique for response time control – brownout – with power capping. We use cascaded control to take into account both the need for predictability in the response times (the inner loop), and the power cap (the outer loop). We execute tests on real machines to determine power usage and response times models and extend an existing simulator. We then evaluate the cascaded controller approach with a variety of workloads and both open- and closed-loop client models.**

I. Introduction

From the beginning of the era of grid and cloud computing, the load pressure on data centers has steadily increased. This is partially due to the increased efficiency obtained by running multiple virtual machines in a single physical machine, often called *server consolidation* [9], [17]. In principle, server consolidation aims to reduce the number of physical machines active and used to serve a given workload through the smart co-location of virtual machines hosting the software applications. Thanks to the reduction in the total number of physical machines active at the same time, it is possible to reach both high utilization and energy efficiency.

However, data center owners face a challenge in providing cost- and energy efficiency while meeting the large variations in resource demands often shown by the applications. To avoid wasting resources, they often apply overbooking techniques in order to cope with the amount of virtual machines that are constantly launched and kept in operation [20]. To deal with some of the drawbacks of overbooking, graceful degradation has been introduced in different forms to be able to scale down the workload when necessary, without the need of firing up a new virtual machine and consuming more physical resources [2], [5], [11], [12], [18].

At the same time, these techniques do not take into account one of the main costs that the data center operator has to face: power. Notably, in addition to the power consumption running costs one should consider that the power delivery infrastructure is one of the most expensive components in a data center. Its cost widely depends on the peak delivery capacity and that capacity is underutilized during most of the operating time.

According to a recent study [8], in a Google data center the normal operating range for power consumption at the cluster level is between 52% and 72% of the peak power, where a cluster comprises about 5000 servers. Scaling down to 800 servers, the range widens to 48–77%, and for a rack of 40 servers the range is even wider, between 45–93%. This means that the data center could potentially host 39% more servers with the same power delivery infrastructure, or it could be built with a less powerful – therefore cheaper – power delivery infrastructure.

This motivates the investigation of this paper into providing means to control the power consumption of a data center, while serving as much as possible the content that the users request. We aim at combining a graceful degradation technique – namely brownout [12] – with power capping. For each virtual machine, the methodology proposed in this paper enforces a dynamically set power cap – the machine cannot consume more than the given amount of power. At the same time, graceful degradation of content helps us maximizing clients satisfaction, given the constraints imposed by the power cap.

The remainder of this paper is organized as follows. Section II presents some background into both graceful degradation and power capping. Section III presents the design of a cascaded controller to obtain the mentioned power capping and user satisfaction goal. Section IV presents some experimental results that validate our proposal and Section V concludes the paper.

II. Background and Related Work

The main idea behind this paper is to combine two different concepts, which have both been explored in the literature, but never jointly. The first concept is *service degradation*, which builds upon the assumption that when the capacity does not allow the server to serve all the users, providing a degraded service to some of the users is better than no service at all. A brief recap on the literature on service degradation is provided in Section II-A. The second concept is called *power budgeting*. Power budgeting was introduced to cap the amount of power consumed by various entities in an

[1] http://www.mercurynews.com/2017/02/28/amazon-cloud-storage-failure-causes-widespread-disruption/

infrastructure, from a single server and a rack, to a cluster and an entire data center. We use power budgeting at the level of the single virtual machine offering an application, distributing therefore the power to the different basic entities in a data center. Section II-B provides a brief overview of the research on power budgeting.

A. Service Degradation

There are situations in which the available computational capacity is not enough to serve the amount of requests that a web server receives. In special conditions, an abrupt increase in the number of hits for a specific page may cause malfunctioning and crushes [1], similar to the Wikipedia and Google News temporal failures caused by Michael Jackson's death in 2009[2].

In these situations it is often better to serve the clients with reduced content (for example eliminating some images or some sections from the response web page) than to not serve the requests at all. In software engineering, this is generally called *graceful degradation*, where a degraded service is offered to the users in place of the original full-quality one. The graceful degradation principle has been applied to various domains [18], from search queries [11], to indexing [5], and database consistency [2].

Recently, the same context has been applied to cloud applications, under the name of Brownout [12], [16], [20]. The idea behind brownout is to produce a response that only contains the minimal components that the user is looking for (the mandatory part) and disable additional computation that would increase the revenue of the application owner (the optional content), like a recommender system that would select similar movies to the one that the user is checking out in an online renting website.

The brownout approach [12] models the 95^{th} percentile of the response times τ_{95} of a cloud application as a function of the percentage of optional content served θ as a first order model with an unknown and possibly time-varying coefficient α that determines the relationship between the two quantities.

$$\tau_{95}(k) = \alpha(k-1)\,\theta(k-1) \tag{1}$$

Brownout uses a controller, designed using loop shaping, to constrain the behavior of the closed loop to be a first order transfer function with static gain one and a pole in p_*. The controller updates the percentage of requests served with optional content $\theta(k)$ based on the error between the response time setpoint $\bar{\tau}_{95}$ and the current measured value. $e_{\tau_{95}}(k) = \bar{\tau}_{95}(k) - \tau_{95}(k)$, and on an estimate $\hat{\alpha}(k)$.

$$\theta^*(k) = \theta(k-1) + \frac{1-p_*}{\hat{\alpha}(k)} e_{\tau_{95}}(k) \tag{2}$$

The value of θ^* is subject to saturation. Since it represents a probability, the following is applied, to compute θ. The value of θ is called *dimmer*, to complete the analogy with electrical brownouts.

$$\theta(k) = \max\{\min\{\theta^*(k), 1\}, 0\} \tag{3}$$

The research work on brownout was then extended to handle multiple servers, with load balancing strategies [6] and to reduce the impact of server failures [13]. With this paper we provide another extension of the original brownout controller, that takes into account power budgeting.

B. Power Budgeting

Power budgeting is an approach to minimize the operational cost of a data center controlling the power usage of all the entities that operate within the data center boundaries. For a data center owner, being prepared to handle a high maximum theoretical power consumption is associated with a substantial cost, especially when the usual power consumption of the data center is much lower than the maximum theoretical value that the owner should provide for. Research on power budgeting has tackled this problem, designing a power delivery infrastructures that is reduced in comparison with the traditional design [8], [15]. Instead of being prepared to handle a peak corresponding to the aggregate peak of all servers, the data center is designed only to provide a usual range of power consumption, and the power that each of the servers can draw from the network is capped to a certain value. In this way, the data center operator can reduce both building and infrastructure costs and operational costs in terms of electricity. Power budgeting exploits (a) knowledge on usual server utilization, (b) the properties of the delivery infrastructure, (c) billing models used by the electricity providers, and (d) the ability of modern machines to throttle their power consumption.

Researchers have tackled the problem of power budgeting at various levels of data center infrastructure: single server [3], [10], [14], [15], [25], rack [19], [21], cluster [23], and whole data center [7], [22], [24]. Generally speaking, power budgeting happens in two different phases: first, power shifting is used to determine the consumption limits assigned to each server. Then, each server individually enforces the power consumption limit using power capping. In this paper we focus only on power capping, since we act at the virtual machine level. We assume that a cap has been determined for each virtual machine and that the virtual machine respects that cap. To this end, there are many actuators, examples being Dynamic Voltage and Frequency Scaling and CPU Pinning. Generally speaking, these actuators reduce the amount of computation that the virtual machine is providing to the connected users, therefore degrading the service.

This paper exploits a graceful degradation technique – brownout – to react to the power capping that is imposed by a central manager with the best delivered performance possible. To the best of the authors knowledge, this is the first research work in which the effects of the power cap are quantified in terms of the amount of performance degradation imposed on the application and on its end users.

[2]http://edition.cnn.com/2009/TECH/06/26/michael.jackson.i nternet/

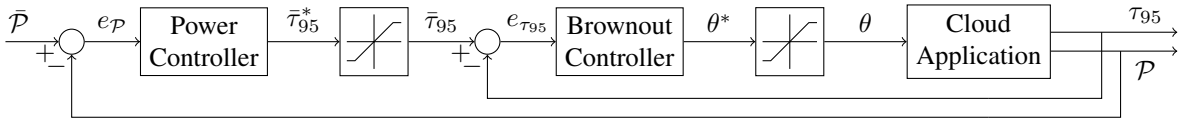

Fig. 1. Block Diagram of the System.

III. Methodology

This paper proposes a cascaded controller with the scheme shown in Figure 1. A power consumption setpoint, $\bar{\mathcal{P}}(k)$ is the input to the system. The setpoint value is compared with the current power consumption of the machine $\mathcal{P}(k)$, computing the error $e_{\mathcal{P}}(k) = \bar{\mathcal{P}}(k) - \mathcal{P}(k)$. The error is the input to a Proportional and Integral (PI) controller, which is in charge of selecting the setpoint for the 95$^{\text{th}}$ percentile in the brownout controller, $\bar{\tau_{95}}$. The setpoint is used to compute the current response time error $e_{\tau_{95}}(k) = \bar{\tau}_{95}(k) - \tau_{95}(k)$. The error is then used by the brownout controller in Equation (2) to compute the dimmer value θ. Both the values of θ and $\bar{\tau}_{95}$ are subject to saturations, as shown in the Figure.

To be precise, the power controller selects the response time setpoint using the following PI controller in velocity form.

$$\bar{\tau}_{95}^{*}(k) = \bar{\tau}_{95}(k-1) + K_p[e_{\mathcal{P}}(k) - e_{\mathcal{P}}(k-1)] + K_i e_{\mathcal{P}}(k) \quad (4)$$

In (4), K_p and K_i are the proportional and integral gain of the controller, respectively -0.1 and -0.02 in our implementation. We also apply saturations, defining as the minimum response time setpoint $\bar{\tau}_{95,\text{min}} = 0.001$ seconds and as the maximum response time setpoint $\bar{\tau}_{95,\text{max}} = 4.0$ seconds, leading to the following control law.

$$\bar{\tau}_{95}(k) = \max\left(\min\left(\bar{\tau}_{95}^{*}(k), \bar{\tau}_{95,\text{max}}\right), \bar{\tau}_{95,\text{min}}\right) \quad (5)$$

Finally, we execute the controller with a period $\tau_c = 0.5s$.

IV. Experimental validation

We validated our controller using the publicly available brownout simulator[3]. The simulator already contains a profile of the amount of time it takes for a server running the brownout-aware RUBiS benchmark[4] to serve requests with and without optional content. Similarly to what the real web server would do, the simulator enqueues the requests and use a process sharing discipline with a small timeslice to simulate the processing capacity given to each of them. In our simulator, clients always behave according to the closed-loop model, i.e., they issue a request, receive a response, wait for a certain think time (3 second), and then issue the following request.

We extended the simulator with power profiles. More precisely, we profiled the real machine obtaining data about the relationship between the 95$^{\text{th}}$ percentile of the response

[3]https://github.com/cloud-control/brownout-simulator
[4]https://github.com/cloud-control/brownout-rubis

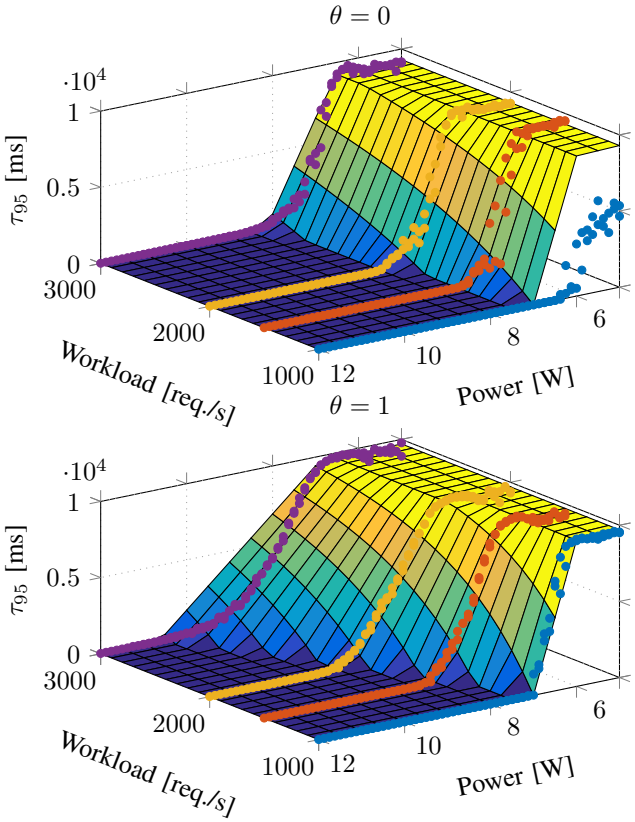

Fig. 2. Power model interpolation.

times τ_{95} and the power consumption $\mathcal{P}$, which we then introduced in the simulator to model the power consumption based on the measured response times. Then we used the enhanced simulator to test the cascaded control loop of Figure 1. In the following, we discuss in details the power profiles and then we delve into the experimental results.

A. Power Profiles

To create power profiles we conducted a set of experiments in which we exposed the RUBiS benchmark to a workload and measured how the application performance changes with the limited power budget. The benchmark was deployed in a virtual machine with 6 virtual CPUs and 8 GB of RAM on a server equipped with Intel Xeon E5-2620 v2 processor. We used multiple experiment settings: (a) serving requests with or without an optional content, (b) limiting the server power budget between 5 and $12W$, and (c) changing the workload level between 1000 and 3000 connected clients. The power budgets were enforced using Running Average Power Limit (RAPL) technology [4]. The measurements are

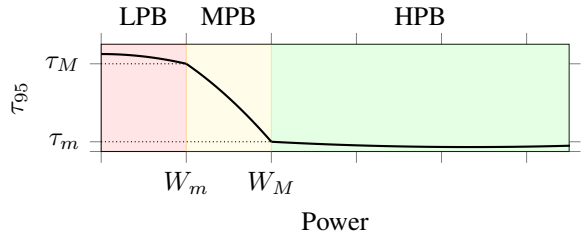

Fig. 3. Power profile regions.

shown in Figure 2, for $\theta = 0$ (top graph), and for $\theta = 1$ (bottom graph).

We analyzed the collected data to identify the relation between the power consumption, the response time, the incoming workload and the value of θ. In particular, we identified a function that maps the current response time, the incoming workload λ and the dimmer value θ into the power consumption.

The power profile can be divided into three different operating regions:

1) **High-Power Budget (HPB)**, where the machine can be operated with sufficient power and guarantees a low response time;
2) **Mid-Power Budget (MPB)**, where the machine is operated with an amount of power that is not able to guarantee reasonable performance, and the response time increases; and
3) **Low-Power Budget (LPB)**, where the machine is operated with the minimum amount of power, i.e., the idle power, and the response time is saturated to the maximum.

Figure 3 shows the typical shape of the power profile described above, evidencing the three different operating regions.

The two highlighted points (W_m, τ_M) and (W_M, τ_m) characterize the transition among the operating regions, and they depend both on the incoming workload, and on the dimmer value. From the experimental data we identified these points for $\theta = 0$ and $\theta = 1$, and for the different workloads, deducing that they can be expressed as:

$$W_m^0(\lambda) = 5 + (\lambda/1000)^{0.4}\,\text{W}, \quad W_M^0(\lambda) = 6 + (\lambda/1000)^{0.7}\,\text{W}$$
$$W_m^1(\lambda) = 5 + (\lambda/1000)^{0.6}\,\text{W}, \quad W_M^1(\lambda) = 6 + (\lambda/1000)^{1.4}\,\text{W}$$
$$\tau_m^0 = 5\text{ms}, \quad \tau_M^0 = 9200\text{ms}$$
$$\tau_m^1 = 7\text{ms}, \quad \tau_M^1 = 9600\text{ms}$$

Therefore, we compute the actual points based on the current workload and dimmer as a convex combination of the obtained points as:

$$W_m(\theta,\lambda) = (1-\theta)W_m^0(\lambda) + \theta W_m^1(\lambda)$$
$$W_M(\theta,\lambda) = (1-\theta)W_M^0(\lambda) + \theta W_M^1(\lambda)$$
$$\tau_m(\theta,\lambda) = (1-\theta)\tau_m^0(\lambda) + \theta \tau_m^1(\lambda)$$
$$\tau_M(\theta,\lambda) = (1-\theta)\tau_M^0(\lambda) + \theta \tau_M^1(\lambda)$$

Finally, we computed the approximated power model as a piecewise affine function, and the result is shown in Figure 2. Notice that the obtained approximation is an invertible function, i.e., given the response time it is possible to compute the power consumption, and vice versa.

B. Experiments

After introducing the power profiles in the simulator, as specified in Section IV-A, we tested the behavior of the cascaded controller when the system is subject to a set of perturbations.

Together with more qualitative considerations, we also evaluate our solution using standard metrics. We compute the Integral of the Absolute Error for both power and response time, respectively indicated with $\mathcal{E}_{\tau_{95}}$ and $\mathcal{E}_{\mathcal{P}}$. Indicating with t the end instant of the simulation, the expressions are computed as follows.

$$\mathcal{E}_{\tau_{95}} = \sum_{k=0}^{t} |e_{\tau_{95}}(k)|, \qquad \mathcal{E}_{\mathcal{P}} = \sum_{k=0}^{t} |e_{\mathcal{P}}(k)| \tag{6}$$

Notice that the data collection is done together with the controller execution, therefore the error sums data points sensed with a time interval of $\tau_c = 0.5s$.

Testing power setpoint variations and incoming request rate changes. Figure 4 shows one of such tests. The simulation here lasts for 500 seconds and is divided into 5 different time intervals, each lasting for 100 seconds. In the interval $[0, 100)$, the application should serve 2000 clients with a power budget of $15W$. In the second interval, between $[100, 200)$ seconds, the application has a power budget of $18W$ and 2000 clients. In between these two intervals only the power budget is changed. The application responds decreasing the setpoint for the response time. Since it has more power to use, it can serve users with a better user experience.

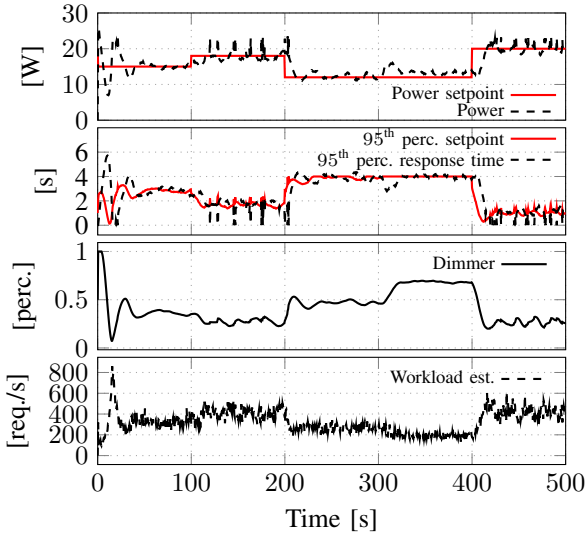

Fig. 4. Closed-loop behavior when the system is subject to power setpoint variations and incoming request rate changes.

In the third interval, when time belongs to $[200, 300)$ the power budget is decreased to $12W$, with 2000 clients. The application then increases the response time setpoint, to keep the users in the system for a longer time and therefore avoid many simultaneous requests. Doing so, the application is able to meet the power budget and also increase the dimmer value. When time equals to 300 seconds, 500 clients disconnects, the load therefore becoming of 1500 concurrent users. It is much easier in this case for the application to meet the power budget and regulate the response time setpoint. The dimmer value can be increased and users receive a better service. Finally, at time 400 and for the last interval, the power budget is increased to $20W$ and the number of clients is increased of 250, making it 1750. As a consequence, the setpoint response time can be decreased, serving the users faster.

The dimmer value is set by the inner loop according to the brownout framework [12], while the last plot shows the workload that is estimated at the application level (the number of simultaneously active requests per second as seen by the application, that is used for the power profiles estimation as described in Section IV-A). Finally, for the experiment shown in Figure 4, $\mathcal{E}_{\tau_{95}} = 511.71$ and $\mathcal{E}_{\mathcal{P}} = 1469.78$.

Testing concurrency effects in the long run. In this second test, we generated traces for a one day long simulation, using patterns of incoming requests that would vary in rate about every 15 minutes (precisely every 1000 seconds). In the simulation, we kept the power budget constant to $15W$, while the number of concurrent users connected to our application was spanning between 750 and 2250. Table I shows the number of connected users for the first 26000 seconds. As can be seen abrupt changes (an increase of 500 users connected or the disconnection of a similar number of users) are tested as well as slow perturbations (both increase and decrease in the order of 50 users).

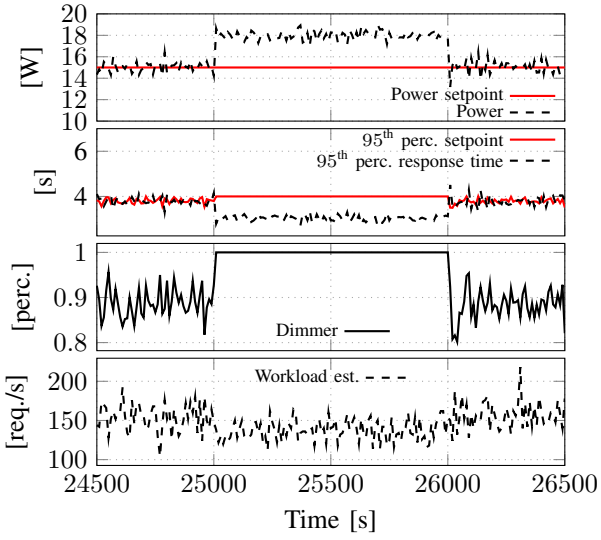

Fig. 5. A close look to a fraction of the day long simulation.

TABLE I
CONNECTED USERS IN THE FIRST PART OF THE SIMULATION.

Interval	Users
[0, 1000)	1500
[1000, 2000)	2000
[2000, 3000)	2250
[3000, 4000)	1750
[4000, 5000)	1250
[5000, 6000)	1000
[6000, 7000)	1250
[7000, 8000)	1750
[8000, 9000)	1250
[9000,10000)	1750
[10000,11000)	1800
[11000,12000)	1850
[12000,13000)	1900
[13000,14000)	1950
[14000,15000)	1450
[15000,16000)	1950
[16000,17000)	1900
[17000,18000)	1850
[18000,19000)	1800
[19000,20000)	1750
[20000,21000)	1450
[21000,22000)	1950
[22000,23000)	2200
[23000,24000)	1700
[24000,25000)	1200
[25000,26000)	950

As a result of the simulation, we analyze both $\mathcal{E}_{\tau_{95}} = 55749.75$ and $\mathcal{E}_{\mathcal{P}} = 130706.81$ more in details. A closer look at the Integral of the Absolute Error of the error in power, reveals that the data corresponds to an average absolute error of $0.756W$ every second (recall that the control period is the same as the data collection period, being $0.5s$). If we count only the samples in which the machine is exceeding the power budget, we obtain an average excess of $0.514W$ per second. At the same time, the Integral of the Absolute Error for the response time reveals an average error of $0.322s$. If we again look only at the excess in absolute time, the average is reduced to $0.120s$, which is acceptable for our application.

The errors are mainly due to periods of time like the one depicted in Figure 5. As can be seen, at time 25000, the system reacts to the incoming request rate change by increasing the response time setpoint to the maximum allowed value imposed by the saturation levels. This in turn leads to an increase in the dimmer value which reaches its own saturation value. At this point, the system is not capable to reduce the consumed power and therefore the consumed power exceeds the setpoint. When these unfeasible conditions do not happen, the controller is capable of regulating the power consumed by the machine.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach to combine the brownout controller for response time control in cloud applications with a power budget controller. The resulting cascaded controller is capable of maintaining a power budget for the running application machine, by using the inner loop

controller to modulate the amount of load that the cloud application receives by changing the response time setpoint.

The idea behind this work is to make cloud applications self-adaptive using a control-theoretical approach and to enforce some guarantees on the resulting closed-loop systems. In this work, it is particularly relevant to notice the effect of the saturation levels for the outer loop, because increased response times come at a cost for the cloud provider, e.g., in terms of reduced costumer retention. Saturating the setpoint for the 95^{th} percentile of the response times at 4 seconds means ensuring that the cloud application is serving clients in the correct way whenever possible and not exceeding some acceptable performance metrics.

We envision extensions of this work that would better exploit the tradeoff between the power budget and the response time. For example, it could be possible to use Model Predictive Control to control the power budget using the response time setpoint as a constraint. At the same time, another future extension of this work would be to include scaling policy to dynamically change the amount of resource given to the cloud application, together with the power budget, thus enforcing the power constraint on a per-core basis.

Acknowledgements: This work was supported by the Wallenberg Autonomous Systems Program (WASP), by the Swedish Research Council (VR) for the projects "Cloud Control", "Feedback Computing" and "Power and temperature control for large-scale computing infrastructures" and by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032.

REFERENCES

[1] A. Ali-Eldin, O. Seleznjev, S. Sjöstedt-de Luna, J. Tordsson, and E. Elmroth. Measuring cloud workload burstiness. In *Proceedings of the 7th International Conference on Utility and Cloud Computing*, UCC '14, pages 566–572. IEEE Computer Society, 2014.

[2] H.-E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez. Harmony: Towards automated self-adaptive consistency in cloud storage. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '12, pages 293–301. IEEE Computer Society, 2012.

[3] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & cap: Adaptive dvfs and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 175–185, 2011.

[4] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194, Aug 2010.

[5] S. Ding, S. Gollapudi, S. Ieong, K. Kenthapadi, and A. Ntoulas. Indexing strategies for graceful degradation of search quality. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 575–584. ACM, 2011.

[6] J. Durango, M. Dellkrantz, M. Maggio, C. Klein, A. V. Papadopoulos, F. Hernández-Rodriguez, E. Elmroth, and K. Årzén. Control-theoretical load-balancing for cloud applications with brownout. In *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, pages 5320–5327, 2014.

[7] M. E. Femal and V. W. Freeh. Boosting data center performance through non-uniform power allocation. In *Proceedings of the Second International Conference on Automatic Computing*, ICAC '05, pages 250–261. IEEE Computer Society, 2005.

[8] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23. ACM, 2007.

[9] P. Graubner, M. Schmidt, and B. Freisleben. Energy-efficient virtual machine consolidation. *IT Professional*, 15(2):28–34, March 2013.

[10] H. Hoffmann and M. Maggio. Pcp: A generalized approach to optimizing performance under power constraints through resource management. In *11th International Conference on Autonomic Computing (ICAC 14)*, pages 241–247. USENIX Association, 2014.

[11] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan. Speeding up distributed request-response workflows. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 219–230. ACM, 2013.

[12] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez. Brownout: Building more robust cloud applications. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 700–711. ACM, 2014.

[13] C. Klein, A. V. Papadopoulos, M. Dellkrantz, J. Dürango, M. Maggio, K.-E. Årzén, F. Hernández-Rodriguez, and E. Elmroth. Improving cloud service resilience using brownout-aware load-balancing. In *Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, SRDS '14, pages 31–40. IEEE Computer Society, 2014.

[14] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura. Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 349–356, Oct 2013.

[15] C. Lefurgy, X. Wang, and M. Ware. Power capping: A prelude to power shifting. *Cluster Computing*, 11(2):183–195, June 2008.

[16] M. Maggio, C. Klein, and K.-E. Årzén. Control strategies for predictable brownouts in cloud computing. *IFAC Proceedings Volumes*, 47(3):689 – 694, 2014. 19th IFAC World Congress.

[17] 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 *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259. ACM, 2011.

[18] J. Philippe, N. De Palma, F. Boyer, and O. Gruber. Self-adapting service level in java enterprise edition. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 8:1–8:20. Springer-Verlag New York, Inc., 2009.

[19] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 66–77. IEEE Computer Society, 2006.

[20] L. Tomás, C. Klein, J. Tordsson, and F. Hernández-Rodríguez. The straw that broke the camel's back: Safe cloud overbooking with application brownout. In *Proceedings of the 2014 International Conference on Cloud and Autonomic Computing*, ICCAC '14, pages 151–160. IEEE Computer Society, 2014.

[21] X. Wang, M. Chen, and X. Fu. Mimo power control for high-density servers in an enclosure. *IEEE Trans. Parallel Distrib. Syst.*, 21(10):1412–1426, Oct. 2010.

[22] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller. Ship: Scalable hierarchical power control for large-scale data centers. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, PACT '09, pages 91–100. IEEE Computer Society, 2009.

[23] X. Wang and Y. Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Trans. Parallel Distrib. Syst.*, 22(2):245–259, Feb. 2011.

[24] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song. Dynamo: Facebook's data center-wide power management system. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 469–480. IEEE Press, 2016.

[25] H. Zhang and H. Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 545–559. ACM, 2016.