A Load-balancer for Brownout-compliant Cloud Applications

AUTHOR1 [*] **AUTHOR2** [*] **AUTHOR3** [**] **AUTHOR4** [*]

[*] *Department of Automatic Control, Lund University*
[**] *Department of Computer Science, Umeå University*

Abstract:

Keywords: Computer systems, Feedback loops, Model-based control, Multiprocessor systems, Probabilistic models, Queuing theory.

1. INTRODUCTION

Cloud computing is expected to be one of the driving force of the future economy (Irwin, 2013). Already, it has dramatically changed the management of computing infrastructures. On one hand, public cloud providers, such as Amazon EC2, allow companies to deploy their services on large infrastructures with no up-front cost (Buyya et al., 2009). On the other hand, the flexibility offered by cloud technologies themselves favors the adoption of private clouds (Gulati et al., 2011), therefore, companies are converting their own computing infrastructures into small, internally-managed clouds.

One of the main advantages offered by cloud infrastructures is **elasticity**. A company that wants to deploy a new service on the cloud can rapidly provision the required computing resources, automatically acquiring and releasing Virtual Machines (VMs) as required (Herbst et al., 2013). Elasticity can be of two complementary types: vertical and horizontal. Vertical elasticity consists in adding or removing resources (e.g., CPU cores) from an existing VM, while horizontal elasticity deals with changing the number of VMs allocated to a specific service, adding a new machine or removing an existing one. Horizontal elasticity calls for the introduction of a specialized component, called **load-balancer**, that takes care of routing the requests to one of the VMs composing the service. Load-balancing techniques have been widely studied and adopted (Barroso and Hölzle, 2009; Lu et al., 2011; Lin et al., 2012).

One of the main issues with cloud computing infrastructures, however, is **robustness** to unexpected events. For example, flash-crowds are sudden increases of end-users, that may increase the required capacity by up to five times (Bodik et al., 2010). Similarly, hardware failures may temporarily reduce the capacity of the data-center, while the failure is repaired (Barroso and Hölzle, 2009). Due to the large magnitude and short duration of such events, it may be economically too costly to provision enough hardware to properly deal with them. As a results, unexpected events may lead to data-center overload, which translates to unresponsive cloud applications, leading to dissatisfied end-users and revenue loss.

In our previous work (Klein et al., 2013), we proposed a cloud application development paradigm called **brownout**. The main characteristic of cloud brownout is to be able to cope with unexpected events by reducing the amount of computation executed inside the VM to produce the response. In brownout applications some computations are marked as optional — for example, the execution of a recommender engine — others are mandatory — the display of the product information in an e-commerce website. Whenever a request is received, the application can choose to execute or not the optional code based on the data-center conditions and on its own optimization. Note that executing optional code directly translates into a better service and more revenues for the company. This approach proved to be successful for dealing with unexpected events. However, we only considered services having a single replica and running inside a single VM.

In this paper, we extend our previous work to multiple VMs, developing a brownout-compliant load-balancer. This enables horizontal elasticity and makes applications fault-tolerant by having several replicas. Existing, state-of-the-art load-balancers forward requests based on the response-time of each replica, which leads to inefficient decisions for brownout-compliant applications, since such applications already keep their response-time at a given setpoint (at the expense of reducing the amount of optional content served). Specifically, a brownout-compliant load balancer is necessary, since 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 threatened with overload executing all optional code, both achieving low response times. Therefore, existing load-balancing techniques that monitor response times should be improved to migrate load away from replicas not executing optional code.

Our challenge is to find a load-balancing methodology that maximizes the amount of served optional content. Our contribution is threefold.

(1) We propose a load-balancing architecture for brownout-compliant cloud applications and formalize our problem (Section 2).
(2) We propose a load-balancing algorithm that maximizes the performance of brownout-compliant applications in terms of percentage of optional code executed (Section 3).
(3) We evaluate the approach showing that it outperforms existing techniques (Section 4).

2. PROBLEM STATEMENT

In this section we present our proposed system architecture and formulate the problem statement.

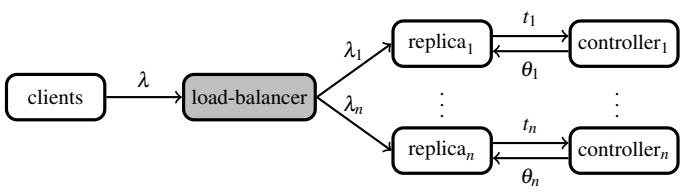

Fig. 1. Architecture of a brownout application featuring multiple replica

Figure 1 presents the architecture of a brownout-compliant application featuring multiple replicas, which reflects established practices in cloud applications Barroso and Hölzle (2009). The main processes are the clients, which are the users of the cloud application, the load-balancer which directs loads towards replicas, the replicas themselves, implementing application logic, and the controllers, which decides the amount of optional content that each replica should serve. Let us describe each process and highlight its specificities related to brownout-compliance.

Clients send requests at a constant rate λ which are send to the load-balancer. The load-balancer forwards these requests to one of the n replicas. As a results, each replica i receives requests with a rate $\lambda_i = w_i \cdot \lambda$, such that $\sum_{i=\overline{1,n}} w_i = 1$, where w_i represents weights computed by the load-balancer.

A replica i may respond to each request in two modes: partially, where only mandatory content is included in the reply, or fully, where only mandatory and optional content is included. The service times for partial and full replies is μ_i and M_i, respectively. Obviously, partial replies are faster to compute than full ones, since the optional content does not need to be computed, hence, $\mu_i > M_i$. Whether the replica serves requests partially or fully depends on a parameter $\theta_i \in [0,1]$, which represents the probability of serving optional content. Assuming the replica is not saturated, it serves requests fully at a rate $\lambda_i \cdot \theta_i$ and partially at a rate $\lambda_i \cdot (1-\theta_i)$. To aid load-balancing decisions, each replica piggy-bags the current value of θ_i through the reply, so that this value can be observed by the load-balancer.

The parameters θ_i are computed by controllers based on a parameter τ, the target average response-time. Each replica i periodically reports the average observed response-time t_i to its controller. In exchange, the controller computes a new value θ_i to close the feedback loop and maintain average response-time.

Given the above architecture, the brownout load-balancing problem can be stated as follows. Assuming the load-balancer can measure λ, t_i and θ_i, compute the values w_i that maximize $\sum_{i=\overline{1,n}} \lambda w_i \theta_i$.

3. SOLUTION

4. EVALUATION

5. RELATED WORK

6. CONCLUSION

ACKNOWLEDGEMENTS

This work was partially supported by the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control and through the LCCC Linnaeus Center. Also, we received partial support from the ELLIIT Excellence Center.

REFERENCES

Barroso, L.A. and Hölzle, U. (2009). *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool.

Bodik, P., Fox, A., Franklin, M.J., Jordan, M.I., and Patterson, D.A. (2010). Characterizing, modeling, and generating workload spikes for stateful services. In *SOCC*, 241–252.

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

Gulati, A., Shanmuganathan, G., Holler, A., and Ahmad, I. (2011). Cloud-scale resource management: challenges and techniques. In *HotCloud*.

Herbst, N.R., Kounev, S., and Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA, June 24–28*.

Irwin, N. (2013). These 12 technologies will drive our economic future. *The Washington Post*. URL `http://goo.gl/EyPBxd`.

Klein, C., Maggio, M., Årzén, K.E., and Hernández-Rodriguez, F. (2013). Introducing service-level awareness in the cloud. Technical Report ISRN LUTFD2/TFRT-7641-SE, Lund University.

Lin, M., Liu, Z., Wierman, A., and Andrew, L.L.H. (2012). Online algorithms for geographical load balancing. In *Proceedings of the 2012 International Green Computing Conference (IGCC)*, IGCC '12, 1–10. IEEE Computer Society, Washington, DC, USA. doi:10.1109/IGCC.2012.6322266. URL `http://dx.doi.org/10.1109/IGCC.2012.6322266`.

Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J.R., and Greenberg, A. (2011). Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Perform. Eval.*, 68(11). doi:10.1016/j.peva.2011.07.015.