Which Cloud Auto-Scaler Should I Use for my Application? Benchmarking Auto-Scaling Algorithms

[Poster Paper]

Ahmed Ali-Eldin
Dept. of Computing Science
Umeå University
ahmeda@cs.umu.se

Alexey Ilyushkin
Delft University of Technology
Delft, the Netherlands
a.s.ilyushkin@tudelft.nl

Bogdan Ghit
Delft University of Technology
Delft, the Netherlands
b.i.ghit@tudelft.nl

Nikolas Roman Herbst
University of Würzburg
Würzburg, Germany
nikolas.herbst@uni-wuerzburg.de

Alessandro Papadopoulos
Dept. of Control
Lund University
alessandro.papadopoulos@control.lth.se

Alexandru Iosup
Delft University of Technology
Delft, the Netherlands
A.Iosup@tudelft.nl

Rapid elasticity is one of the essential characteristics of cloud computing identified by NIST [17]. Elasticity allows resources to be provisioned and released to scale rapidly out ward and in ward according to demand. Tens – if not hundreds – of algorithms have been proposed in the literature to automatically achieve elastic provisioning [15, 23, 14, 21, 13, 20, 6, 12, 16, 10]. These algorithms are typically referred to as elasticity algorithms, dynamic provisioning techniques or autoscalers.

While trying to solve the same problem, sometimes with differing assumption, many of these algorithms are either compared to static provisioning or to a predefined QoS target, e.g., predefined response time target, with very little – or no – comparison to previously published work. This reduces the ability of an application owner or a cloud operator to choose and deploy a suitable algorithm from the literature. Many of these algorithms have been tested with one single – real or synthetic – workload in a specific use-case [13, 14, 10]. While all published algorithms are shown to work in the specific use-case they were designed for with the , typically short, workloads tested with, it is seldom the case that the real scenarios will be any thing close to the test cases for which the algorithms are shown to work. Bursts occur in workloads occasionally. Workload dynamics change over time and the load-mix of an application significantly affects how provisioning should be done [21].

This work aims to validate and better understand the literature on automated rapid elasticity algorithms under different conditions. We compare 10 auto-scalers from the state-of-the-art, namely, [23, 7, 14, 6, 5, 12, 18, 4, 13, 11, 10, 8]. We obtained the code for five of the auto-scalers from their designers and reimplemented five.

Since cloud applications are heterogeneous in nature with different resource requirements and workload characteristics [19], for this study, we choose a set of representative applications. Each of these applications stress different resources. The set of applications chosen include complex webservices, scientific workflows, big data processing, simple web services, and video streaming, Due to lack of space, we describe only two applications in more details.

1. Scientific applications and workflows. Currently, workflows are widely used to drive complex computations. A *workflow* (WF) or a *Directed Acyclic Graph* (DAG) consists of a set of *tasks* (nodes) which have precedence constraints among them. Any task can start the execution when all of its input dependencies are satisfied. The whole workflow in our setup is considered as a *job*. The popularity of workflows brings the diversity of their structures, sizes, and resource requirements. For our experiments we use three well known scientific workflows, namely, Montage, LIGO, and SIPHT. Additionally, we consider two generic workflow structures: a star (bag-of-tasks) and a chain. To schedule and execute workflows we use the KOALA scheduler [9] and OpenNebula private cloud which are deployed on the DAS-4 infrastructure [1]. The execution environment for workflow tasks consist of a single head VM and multiple worker VMs.

2. Complex Web-Services. Wikipedia, the free online encyclopedia, is one of the top 10 accessed websites on the Internet [2]. The Wikimedia foundation has open sourced MediaWiki, a custom-made, free and open-source wiki software platform written in PHP and JavaScript. MediaWiki runs using a LAMP stack (Linux, Apache, MySql, and PHP) or similar installations. This setup is similar to many cloud applications [22] including, e.g., Facebook which uses a modern version of the LAMP stack, and YouTube [1]. In our experiments, we replicate the German Wikipedia on DAS. We have chosen the German Wikipedia since it is one of the most popular Wikis in terms of number of users, and in terms of number of articles [3]. The Wikipedia server architecture deployed in their datacenters is multi-tiered with each tier deployed on a different set of servers, but in our setup, we reduce the number of layers to reduce the complexity. We have chosen to run a bare minimum setup, i.e., with the MediaWiki software, a load-balancer and a MySQL database.

[1] www.cs.vu.nl/das4

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICPE'16 March 12-18, 2016, Delft, Netherlands

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4080-9/16/03.
DOI: http://dx.doi.org/10.1145/2851553.2858677

We benchmark the performance of the selected auto-scalers with the chosen applications and quantify the performance based on the following set of metrics.

1. Average Overprovisioning ($\overline{OP}$) is the average number of overprovisioned VMs by the autoscaler per unit time. It is calculated by summing the number of overprovisioned VMs over time (OP) and dividing the number by the total time for which the autoscaler was running. A machine is considered overprovisioned if it is of no use for the next 10 minutes. This time window reduces the penalty if an algorithm predicts the future workload well in advance.

2. Average Underprovisioning ($\overline{UP}$) is the average number of underprovisioned VMs by the autoscaler per unit time. It is calculated by summing the number of underprovisioned VMs over time (UP) and dividing the number by the total time for which the autoscaler was running. Underprovisioning means that the autoscaler failed to provision the resources required to serve all requests on time.

3. Average number of Oscillations ($\overline{O}$) which is the average number of VMs started or shut-down (O) per unit time. The reason to consider ($\overline{O}$) as an important parameter is the cost of starting/stopping a VM. From our experience, starting a machine (physical or virtual) takes from one minute up to several minutes depending on the application running (almost 20 minutes for an ERP application server). This time does not include the time required to transfer the images and any data needed but is rather the time for (virtual) machine boot, network setup and application initiation. Similar time may be required when a machine is shutdown for workload migration and load balancer reconfiguration.

4. Maximum, minimum and, average time required for computing the prediction ($\overline{T}$). When possible, we also report the computational complexity of each algorithm. These values do not change significantly for the same algorithm with respect to the application managed. We thus report these values for all our experiments and comment on any anomalies in the measured times between experiments.

In addition to these general metrics, we look at application specific metrics for the applications such as response time, throughput and request drop rate for the web services, queue length for the scientific workflows and the big data workload, and jitter in the video streaming workload, The aim of our work is to provide researchers and industries with a better understanding of the current state-of-the-art.[2]

1. REFERENCES

[1] 7 Years Of YouTube Scalability Lessons In 30 Minutes. Accessed: October, 2015, URL: http://highscalability.com/blog/2012/3/26/7-years-of-youtube-scalability-lessons-in-30-minutes.html.

[2] Top Sites. Accessed: October, 2015, URL: http://www.alexa.com/topsites.

[3] Wikipedia Statistics. Accessed: October, 2015, URL: https://stats.wikimedia.org/EN/Sitemap.htm.

[4] A. Al-Shishtawy and V. Vlassov. ElastMan: Autonomic elasticity manager for cloud-based key-value stores. In *HPDC 13*, pages 115–116, 2013.

[2]This project is done as a collaboration between different partners in the SPEC Cloud Research Group.

[5] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *ScienceCloud*, pages 31–40, 2012.

[6] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *NOMS 12*, pages 204–212, April 2012.

[7] T. Chieu, A. Mohindra, A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *ICEBE 09*, pages 281–286, Oct 2009.

[8] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[9] L. Fei, B. Ghit, A. Iosup, and D. Epema. KOALA-C: A task allocator for integrated multicluster and multicloud environments. In *CLUSTER*, pages 57–65, 2014.

[10] H. Fernandez, G. Pierre, and T. Kielmann. Autoscaling web applications in heterogeneous cloud infrastructures. In *IC2E*, 2014.

[11] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang. Adaptive, model-driven autoscaling for cloud applications. In *ICAC*, pages 57–64, 2014.

[12] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. AutoScale: Dynamic, robust capacity management for multi-tier data centers. *TOCS*, 30(4):14:1–14:26, Nov. 2012.

[13] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *ICPE*, pages 187–198, 2013.

[14] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *FGCS*, 27(6):871–879, 2011.

[15] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh. Automated control in cloud computing: Challenges and opportunities. In *ACDC*, pages 13–18, 2009.

[16] A. H. Mahmud, Y. He, and S. Ren. Bats: Budget-constrained autoscaling for cloud performance optimization. *SIGMETRICS*, 42(1):563–564, June 2014.

[17] P. Mell and T. Grance. The nist definition of cloud computing. 2011.

[18] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. AGILE: Elastic distributed resource scaling for Infrastructure-as-a-Service. In *Proc. 10th Int. Conf. on Autonomic Computing*, ICAC 13, pages 69–82, 2013.

[19] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *SoCC*, page 7. ACM, 2012.

[20] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. CloudScale: Elastic resource scaling for multi-tenant cloud systems. In *SOCC*, pages 5:1–5:14, 2011.

[21] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *ICAC*, pages 21–30, 2010.

[22] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai. Are clouds ready for large distributed applications? *ACM SIGOPS Operating Systems Review*, 44(2):18–23, 2010.

[23] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM TAAS*, 3(1):1:1–1:39, 2008.