IaaS Reactive Autoscaling Performance Challenges

Vladimir Podolskiy, Anshul Jindal and Michael Gerndt
Chair for Computer Architecture and Parallel Systems
Technical University of Munich
Garching (near Munich), Germany
Email: v.podolskiy@tum.de, anshul.jindal@tum.de, gerndt@in.tum.de

***Abstract*—The main feature of a cloud application is its scalability. Major IaaS cloud services providers (CSP) employ autoscaling on the level of virtual machines (VM). Other virtualization solutions (e.g. containers, pods) can also scale. An application scales in response to change in observed metrics, e.g. in CPU utilization. Occasionally, cloud applications exhibit the inability to meet the Quality of Service (QoS) requirements during the scaling caused by the reactivity of autoscaling solutions. This paper provides the results of the autoscaling performance evaluation for two-layered virtualization (VMs and pods) conducted in the public clouds of AWS, Microsoft and Google using the approach and the Autoscaling Performance Measurement Tool developed by the authors [5], [6].**

***Keywords*-autoscaling performance; autoscaling; multilayered autoscaling; cloud computing**

I. Introduction

Scalability became the main feature of cloud infrastructure and services [10], [8]. As the customer base tends to change rapidly, the once perfectly fitting data center becomes obsolete in an instant. IaaS *autoscaling* technology allows to dynamically adjust the number of VMs as long as there is free hardware capacity left in the cloud [4]: if a webshop hosted in the cloud experiences an increase in requests, additional VMs could be provided to cope with the load; vice-versa, the VM instances could also be automatically terminated in case of traffic decrease. Currently, autoscaling is used to find a balance between providing high quality services and minimizing the costs induced by cloud usage.

IaaS autoscaling solutions adjust the virtualized resources in response to a changing demand resulting in the changing virtualized resources utilization. Such solutions as Kubernetes and Docker Swarm support the autoscaling on the level of application services. Thus, the number of service instances is adapted to fit the demand and is balanced across the provided hardware resources and/or VMs. Despite the differences in the approaches to virtualization, the autoscaling solutions share the common reactive approach to autoscaling of the virtualized resources and services.

The goal of the research was to identify whether the reactive nature of autoscaling solutions jeopardizes the ability of cloud applications to meet the QoS requirements under the dynamically changing load. To evaluate the reactive autoscaling solutions, we have tested combinations of AWS Auto Scaling, Azure Autoscale, Google Compute Engine Autoscaling with Kubernetes horizontal scaling of pods. The experiments were conducted using the Autoscaling Performance Measurement Tool (APMT) developed by one of the authors [6]. The approach originally presented in the paper [5] was used to evaluate the performance of the autoscaling solutions.

The comparison of autoscaling solutions and the discussion is provided in the following section. The third section contains the overview of the related works. The last section concludes the paper and provides future research directions.

II. Comparison of the Autoscaling Solutions

A. Evaluation Tool

For the purpose of evaluation, the Autoscaling Performance Measurement Tool was employed. APMT collects the performance data (latency, and number of failed requests) for several IaaS autoscaling solutions, currently including AWS Auto Scaling, Azure Autoscale, and Google Compute Engine Autoscaling. The support for Kubernetes horizontal scaling of pods is also included to enable the evaluation of the autoscaling performance for the multilayered cloud applications. The description of the tool is not provided due to space limitations, it could be found in [6].

B. Experimental Setting

All four workload patterns currently supported by APMT were used in the tests: linear increase, linear increase and constant, random, and triangle. The total time for each test was 20 minutes; request timeout was kept to 6.5 seconds. The number of simulated concurrent clients was 50, they were deployed on the single VM instance not participating in the experiment. For each pattern (except for random) the start value of requests rate was 1, whereas the increase/decrease step was set to be 3. The random load pattern starts at 50 requests and increases/decreases randomly. The requests load generation was uniformly distributed among the concurrent clients. The test application (compute-intensive) computed the sum of prime numbers between 1 and 1000000 when called.

The VM configuration is provided in Table I.

Table II depicts the configuration of Kubernetes autoscaling solution. Table III contains the configuration settings for CSPs native autoscalers.

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00144

Authorized licensed use limited to: Imperial College London. Downloaded on January 11,2024 at 20:04:43 UTC from IEEE Xplore. Restrictions apply.

Table I
EXPERIMENTAL VM CONFIGURATION

CSP	Instance type	Memory	vCPUs	OS Image
AWS	t2.small	2 GB	1 vCPU	Ubuntu 16.04
Microsoft	A1_V2 Std.	2 GB	1 vCPU	Ubuntu 16.04
Google	-	2 GB	1 vCPU	Ubuntu 16.04

Table II
EXPERIMENTAL CONFIGURATION: KUBERNETES AUTOSCALER

Instances	Min. pods	Max. pods	Scaling metric	*Threshold*
1(master) 3(minions)	1	10	CPU Utilization	20 %

C. Experimental Results

The experiment was conducted several times for each combination of autoscaling solutions. As the results demonstrated relative stability in performance and scaling patterns and we wanted to highlight autoscaling behavioral features that might be lost by averaging, we have chosen the results of the *single* experiment. For the sake of brevity, the graphs are only provided for the single load pattern - linear load increase followed by the constant value of the load.

1) AWS Auto Scaling + Kubernetes: The data collected in the scope of AWS Auto Scaling/Kubernetes experiment demonstrate that the scale-out action conducted by the native AWS autoscaling solution lags the scale-out action by Kubernetes which results in the deployment of new pods on a single VM (see rows **B**, **C** of column 1 in Fig. 1). This behavior indicates potential coordination problems between multiple virtualization layers. The lack of coordination could lead to the deployment of new pods on the old VM instances set during the scale-out, whereas the newly added VM could only have single pod. Such a disproportion could lead to load balancing issues and may result in the latency increase as is shown in row **D** of the first column. The scale-in times for AWS Auto Scaling are larger than scale-out times which is indicated by the plot **C-1**. A possible explanation is that AWS Auto Scaling conducts more time-consuming actions during the termination of the VM. For the linear increase and random load patterns not presented in the figure, the current number of pods was reduced although Kubernetes did not request this reduction. The reason is that the infrastructure scaling decided to decommission VMs although pods were still running there.

2) Microsoft Azure Autoscale + Kubernetes: Microsoft Azure Autoscale demonstrates the slowest autoscaling behavior (refer to plot **C-2** in Fig. 1). Both scale-out and scale-in times are significantly larger than for GCE and AWS for all the load patterns tested. It is possible to note in the Azure graphs in rows **B-E** that the performance heavily relies on the underlying hardware and on the scaling of Kubernetes pods, and not on the actual number of VMs. This behavior is even more evident for the other three tested load patterns, excluded from the figure for the sake of brevity.

3) Google Compute Engine (GCE) autoscaling + Kubernetes: Rows **D-E** in column 3 of Fig. 1 show that the GCE/Kubernetes installation exhibits the best performance among tested configurations. Looking at the plot **C-3**, we can see a cause for such a behavior - the most part of the experiment interval is covered by the scaled-out VM instances. If pod replicas are distributed over more VMs, they can take a higher load. However, there is always a trade-off between the size of VMs, their number and the number of pod replicas. For example, early VMs scale-out could result in the cost increase. The experiments also show that GCE autoscaling is faster at taking scaling decision and providing the VM instances than AWS and Azure. Additionally, the best coordination between the native autoscaling solution and Kubernetes autoscaling is exposed by GCE/Kubernetes installation as illustrated by rows **B-C** in column 3 - new pods are mostly added after the new VM instance was added.

4) Autoscaling Solutions Comparison: The comparison of the AWS, Azure, and GCE installations is conducted using two metrics: 1) the amount of QoS violations; 2) the fractions of the autoscaling intervals where the QoS requirements were violated. In the Table IV the number of QoS violations by load pattern are summarized. A violation of the latency QoS requirement is identified by the mean latency being higher than 2.5 seconds. Errors QoS violation is indicated by the amount of errors being higher than 10. Table V provides performance evaluation during the autoscaling events using the method described in [5].

The results in the Table IV show that GCE/Kubernetes installation outperforms AWS and Azure. The initial cause is the fast decision-making process for VMs instances scale-out coupled with the synchronization of the autoscaling on VMs and pods layers. However, in respect to the amount of requests ending up in error, results show no clear leader.

GCE/Kubernetes installation shows problems handling linear increase and random load patterns. If we refer to the errors plot **E-3** in Fig. 1, we might notice really small intervals with high amount of errors. Turns out that GCE/Kubernetes solution exhibits a denser structure of the responses ending up in error codes returned. For the sake of brevity, we do not provide the zoomed version of this plot.

The Table V summarizes parameters of all CSPs layer *scale-out intervals*. Column *HL* represents a fraction of the autoscaling interval with the violated mean latency QoS, whereas *FR* represents the same for the number of requests ending with time-out error. As not all the installations have exposed the clear synchronized multilayered behavior, we

Table III
EXPERIMENTAL CONFIGURATION: CSPS AUTOSCALER

Min. instances	Max. instances	Scaling metric	*Threshold*
1	3	CPU Utilization	20 %

Authorized licensed use limited to: Imperial College London. Downloaded on January 11,2024 at 20:04:43 UTC from IEEE Xplore. Restrictions apply.

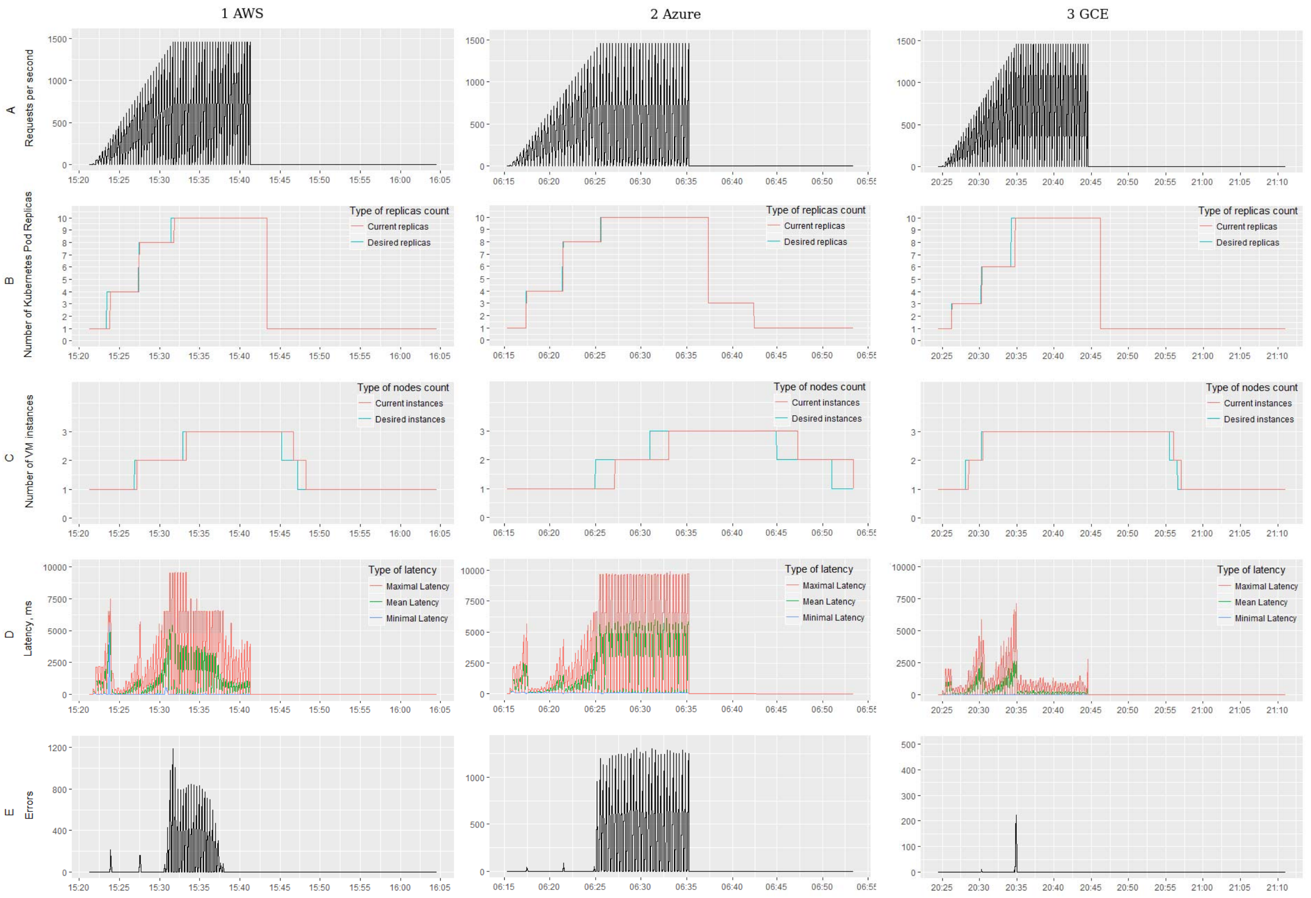

Figure 1. The results of the multilayered autoscaling evaluation for the linear increase and constant pattern. Rows: A) Number of requests sent; B) Desired and current amount of Kubernetes pods; C) Desired and current amount of VM instances; D) Minimal, Mean, and Maximal latency; E) Number of errors.

Table IV
PERFORMANCE COMPARISON BASED ON THE AMOUNT OF QOS VIOLATIONS

Load Pattern	**Amount of latency breaks**			**Amount of errors breaks**		
	AWS	Azure	GCE	AWS	Azure	GCE
Linear Increase	0	0	0	**0**	**0**	382
Linear and Constant	17962	40934	**250**	25707	42725	**1251**
Random	1545	2570	**1127**	**1720**	2570	1835
Triangle	6418	15222	**76**	9954	16368	**845**

have evaluated autoscaling performance during the scaling of VMs.

Results shown in Table V indicate that autoscaling may result in the performance issues and QoS requirements violations. Making scaling intervals smaller and loosening thresholds in the autoscaling rules does not necessarily help to increase the performance of the installation during the autoscaling as the old infrastructure remains exposed to the arriving requests. We can also observe a clear autoscaling performance problem for Azure. Scale-out times of other installations for all the patterns are mostly in the 5 - 30 seconds interval which could be considered appropriate.

III. RELATED WORKS

The principles of the autoscaling policies performance evaluation were introduced by A. Papadopoulos et al. [9]. The performance estimation approach presented by A. Evangelidis et al. [1] is based on probabilistic discrete-time Markov chains models checking. The major contribution of the study by A. Ilyushkin et. al. [3] is a set of performance metrics to estimate each autoscaling policy. The technical report [7] by L. Versluis et al. highlights that the application domain actually heavily influences the performance of the autoscaler. K. Hwang et al. have outlined the generic performance model for clouds of any type with a total of 19 metrics divided into 3 abstraction levels: basic performance metrics, cloud capabilities, cloud productivity [2].

Authorized licensed use limited to: Imperial College London. Downloaded on January 11,2024 at 20:04:43 UTC from IEEE Xplore. Restrictions apply.

Table V
PERFORMANCE OF AWS, AZURE, AND GCE AUTOSCALING SOLUTIONS FOR SCALE-OUT EVENTS

Load Pattern	Installation	Scale-out	Atoscaling Time, *seconds*	HL	FR
Linear Increase	AWS	1^{st}	28.06	0.00	0.00
		2^{nd}	9.03	0.00	0.00
	Azure	1^{st}	**128.00**	0.00	0.00
		2^{nd}	126.00	0.00	0.00
	GCE	1^{st}	8.01	0.00	0.00
		2^{nd}	12.01	0.00	0.00
Linear Increase and Constant	AWS	1^{st}	17.03	0.00	0.00
		2^{nd}	28.08	0.73	0.88
	Azure	1^{st}	**128.00**	0.85	0.92
		2^{nd}	123.00	**0.92**	**0.94**
	GCE	1^{st}	26.02	0.00	0.00
		2^{nd}	11.95	0.02	0.02
Random	AWS	1^{st}	33.08	0.00	0.00
		2^{nd}	15.01	0.00	0.00
	Azure	1^{st}	**131.00**	0.00	0.00
		2^{nd}	117.00	0.00	0.00
	GCE	1^{st}	11.01	0.00	**1.00**
		2^{nd}	7.99	0.00	0.00
Triangle	AWS	1^{st}	6.01	0.00	0.00
		2^{nd}	18.02	0.00	0.00
	Azure	1^{st}	**155.00**	**0.86**	**0.91**
		2^{nd}	128.00	0.00	0.00
	GCE	1^{st}	7.98	0.00	0.00
		2^{nd}	8.01	0.00	0.00

IV. CONCLUSION

The results of the conducted comparison show that a real impact on the performance characteristics of multilayered cloud applications could be produced by the time that decision to scale takes, as well as by the real hardware underlying VM instances, and by the degree of synchronization between autoscaling on different virtualization layers. In the untuned case, GCE/Kubernetes solution shows the best overall performance which could be attributed to the over-provisioning of VMs.

The study has indicated several future research directions: 1) dimension of scaling - to identify when to scale the number of VMs or change the type of VMs used in Kubernetes cluster; 2) intelligent cross-layer policies - to identify which information on Kubernetes level could help in taking the decision on the infrastructure level; 3) interaction of autoscaling decisions for different services - to synchronize the real applications scaling decisions over multiple services of the application.

The performance pitfalls follow the reactive nature of the autoscaling solutions. In order to avoid these pitfalls, the predictive autoscaling techniques might be used. Using the forecasting models for load prediction, the performance models of the virtual entities to determine the load that could be taken by the virtual entity instance without violating the QoS requirements, one can plan the scaling actions in advance and thus overcome the highlighted scalability performance issues.

REFERENCES

[1] Alexandros Evangelidis, David Parker, and Rami Bahsoon. Performance modelling and verification of cloud-based auto-scaling policies. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '17, pages 355–364, Piscataway, NJ, USA, 2017. IEEE Press.

[2] K. Hwang, X. Bai, Y. Shi, M. Li, W. G. Chen, and Y. Wu. Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):130–143, Jan 2016.

[3] Alexey Ilyushkin, Ahmed Ali-Eldin, Nikolas Herbst, Alessandro V. Papadopoulos, Bogdan Ghit, Dick Epema, and Alexandru Iosup. An experimental performance evaluation of autoscaling policies for complex workflows. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ICPE '17, pages 75–86, New York, NY, USA, 2017. ACM.

[4] D. Jayasinghe, S. Malkowski, J. Li, Q. Wang, Z. Wang, and C. Pu. Variations in performance and scalability: An experimental study in iaas clouds using multi-tier workloads. *IEEE Transactions on Services Computing*, 7(2):293–306, April 2014.

[5] A. Jindal, V. Podolskiy, and M. Gerndt. Multilayered cloud applications autoscaling performance estimation. In *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, pages 24–31, Nov 2017.

[6] Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Autoscaling performance measurement tool. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 91–92, New York, NY, USA, 2018. ACM.

[7] A. Iosup L. Versluis, M. Neacsu. Technical Report: A Trace-Based Performance Study of Autoscaling Workloads of Workflows in Datacenters. TR 1711.08993v1, Vrije Universiteit Amsterdam, nov 2017.

[8] D. Moldovan, H. L. Truong, and S. Dustdar. Cost-aware scalability of applications in public clouds. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 79–88, April 2016.

[9] Alessandro Vittorio Papadopoulos, Ahmed Ali-Eldin, Karl-Erik Arzen, Johan Tordsson, and Erik Elmroth. Peas: A performance evaluation framework for auto-scaling strategies in cloud applications. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 1(4):15:1–15:31, August 2016.

[10] N. Serrano, G. Gallardo, and J. Hernantes. Infrastructure as a service and cloud technologies. *IEEE Software*, 32(2):30–36, Mar 2015.

Authorized licensed use limited to: Imperial College London. Downloaded on January 11,2024 at 20:04:43 UTC from IEEE Xplore. Restrictions apply.