Coordinating Vertical Elasticity of both Containers and Virtual Machines

Yahya Al-Dhuraibi[1,2], Faiez Zalila[1], Nabil Djarallah[2] and Philippe Merle[1]
[1]*Inria / University of Lille, France*
[2]*Scalair Company, France*

Keywords: Cloud Computing, Container, Docker, Vertical Elasticity.

Abstract: Elasticity is a key feature in cloud computing as it enables the automatic and timely provisioning and deprovisioning of computing resources. To achieve elasticity, clouds rely on virtualization techniques including Virtual Machines (VMs) and containers. While many studies address the vertical elasticity of VMs and other few works handle vertical elasticity of containers, no work manages the coordination between these two vertical elasticities. In this paper, we present the first approach to coordinate vertical elasticity of both VMs and containers. We propose an auto-scaling technique that allows containerized applications to adjust their resources at both container and VM levels. This work has been evaluated and validated using the RUBiS benchmark application. The results show that our approach reacts quickly and improves application performance. Our coordinated elastic controller outperforms container vertical elasticity controller by *18.34%* and VM vertical elasticity controller by *70%*. It also outperforms container horizontal elasticity by *39.6%*.

1 INTRODUCTION

Cloud computing is an attractive paradigm to many application domains in industry and academia. An enormous number of applications are deployed on cloud infrastructures. The workload of cloud applications usually varies drastically over time. Therefore, maintaining sufficient resources to meet peak requirements can be costly, and will increase the application provider's functional cost. Conversely, if providers cut the costs by maintaining only a minimum computing resources, there will not be sufficient resources to meet peak requirements and cause bad performance, violating Quality of Service (QoS) and Service Level Agreement (SLA) constraints. Cloud elasticity takes an important role to handle such obstacle. Cloud elasticity is a unique feature of cloud environments, which allows to provision/deprovision or reconfigure cloud resources, i.e., Virtual Machines (VMs) and containers (Al-Dhuraibi et al., 2017b). Cloud elasticity can be accomplished by horizontal or vertical scaling. Horizontal elasticity consists in adding or removing instances of computing resources associated to an application (Coutinho et al., 2015). Vertical elasticity consists in increasing or decreasing characteristics of computing resources, such as CPU, memory, etc. (Lakew et al., 2014).

VMs and containers are the main computing resource units in cloud computing. VMs are the traditional core virtualization construct of clouds. Containers are a new competitor, yet complementary virtualization technology to VMs. In this paper, we use Docker containers. Docker[1], a recent container technology, is a system-level virtualization solution that allows packaging an application with all of its dependencies into standardized units for software deployment. While VMs are ultimately the medium to provision PaaS and application components at the infrastructure layer, containers appear as a more suitable technology for application packaging and management in PaaS clouds (Pahl, 2015). Containers can run on VMs or on bare OS. Running containers or different containerized applications in VM or cluster of VMs is an emerging architecture used by the cloud providers such as AWS EC2 Container Service (ECS), Google Cloud Platform, MS Containers, Rackspace, etc. The VMs are run by the hypervisors on the host. Our work manages resources for such architecture. Therefore, this paper addresses the combination of vertical elasticity of containers and vertical elasticity of VMs.

Many works (Baruchi and Midorikawa, 2011), (Dawoud et al., 2012), (Farokhi et al., 2015) handle the vertical elasticity of VMs, other works (Monsalve et al., 2015), (Paraiso et al., 2016) manage the resources of containers. However, no

[1]http://docker.io

Al-Dhuraibi, Y., Zalila, F., Djarallah, N. and Merle, P.
Coordinating Vertical Elasticity of both Containers and Virtual Machines.
DOI: 10.5220/0006652403220329
In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, pages 322-329
ISBN: 978-989-758-295-0

attention was given to the coordination of both elasticities. In addition, these works do not take in consideration that the added resources to the VM are not detected by Docker daemons. We propose the first system powering the coordination between VM and container vertical elasticity. In this paper, we propose a controller coordinating container vertical elasticity with the hosting VM vertical elasticity. This approach autonomously adjusts container resources according to workload demand. Subsequently, we control VM resources if the hosted containers require more resources. Docker daemon does not automatically detect the "on-the-fly" or hot added resources at the VM level unless it is reinstalled or its dedicated cgroups modification. Our system enables Docker daemons to detect the added resources to the hosting VM, therefore, containers can make use of these resources. In addition to the scientific aspects listed in the below items, we have added this technical contribution. Our approach is evaluated by conducting experiments on the benchmarking application RUBiS (Cecchet et al., 2002). RUBiS is an implementation of an auction site similar to eBay, it is widely used in clouds to evaluate J2EE application servers performance and scalability. The results show that our approach improves application performance. It outperforms container vertical elasticity controller by 18.34% and VM vertical elasticity controller by 70%, it also outperforms container horizontal elasticity by 39.6%. The main contributions of this paper are:

1. An autonomous vertical elasticity system for both Docker containers and the hosting VMs. It allows to add/remove resources (i.e., CPU cores, memory) according to the workload demand.
2. We show that our combination of vertical elasticity of both VMs and containers is better than the vertical elasticity of VM only or the vertical elasticity of containers only, i.e., $V_{vm}.V_{cont} > V_{vm} \oplus V_{cont}$, where (.) and ($\oplus$) are the symbols for the logical *AND* and *XOR* operators, respectively. V_{vm} denotes the vertical elasticity of VMs while V_{cont} denotes the vertical elasticity of containers.
3. We show that our combination of vertical elasticity of both VMs and containers is better than the horizontal elasticity of containers, i.e., $V_{vm}.V_{cont} > H_{cont}$, where H_{cont} denotes the horizontal elasticity of containers.

The rest of this paper is organized as follows. Section 2 discusses the motivation towards this work. Section 3 describes the design and function of our coordinating vertical elasticity controller system. Section 4 presents the evaluation of our solution. In Section 5 we discuss some related works. Section 6 presents conclusion and future work.

2 MOTIVATION

There is a large amount of research on cloud elasticity, however, most of them are based on VMs. Some works highlight elasticity of containers and they are discussed in Section 6. With the varying application workload demand, container(s) on the host continues to scale up/down resources, thanks to our Docker controller which manages container resource allocation. The problem is that when containers have already allocated all resources from the host machine, the containerized application performance will be degraded. Therefore, to handle such obstacle and to add more resources, one of the following mechanisms should be used: horizontal elasticity, migration, or reconfiguration of the host machine (i.e., vertical elasticity). We experiment these mechanisms and show that the vertical elasticity is better when it is possible in terms of performance and configurations.

Horizontal Elasticity: Since horizontal elasticity consists in replicating the application on different machines, some applications such as vSphere and DataCore require additional licenses for each replica. These licenses could be very expensive. Besides, horizontal elasticity requires additional components such as load balancers and their reconfiguration. The initialization of an instance takes also a time during the boot process to be functional. These requirements are not needed for the vertical elasticity. In (Dawoud et al., 2012), they have mathematically and experimentally proved that the vertical elasticity is better than the horizontal elasticity. To verify this fact, they have used queuing theory (Sztrik, 2012). Although horizontal elasticity has many advantages including redundancy, being able to scale to almost any scale, and allowing load balancing over multiple physical machines, it requires additional components and reconfiguration. (Appuswamy et al., 2013) proves that vertical elasticity outperforms horizontal scaling in terms of performance, power, cost, and server density in the world of analytics, mainly in Hadoop MapReduc. In addition, horizontal elasticity is not a good choice for the stateful applications that require sticky sessions. Finally, coordinated vertical scaling is desired when there is enough capacity of the physical servers, horizontal scalability may still be needed, since vertical scalability is ultimately limited by the capacity of resources.

Migration: the other choice to have more resources is to migrate the container to another machine with more resources. To experiment this mechanism, we implement live migration technique for Docker containers. CRIU (Checkpoint/Restore, 2017) is used to achieve the procedure and migrate containers lively (Al-Dhuraibi et al., 2017a). CRIU is a Linux functionality that allows to checkpoint/restore processes, e.g., Docker containers. CRIU has the ability to save the state of a running process so that it can later resume its execution from the time of the checkpoint. We take many pre-dumps of the container while it is running, then a final dump for the memory page changes after the last pre-dump is taken (this time the Docker container freezes). While an efficient mechanism is used to transfer Docker containers, there is still downtime due to the migration when the container process is frozen. Table 1 shows migration down time for two small size applications (nginx, httpd). Network traffic overhead is not considered. Container migration is also risky for stateful applications such RTPM media applications to lose sessions.

Table 1: Migration Performance Indicators.

App.	Image size (MB)	Pre-dump time (s)	Dump time (s)	Restore time (s)	Migr. total time (s)	Migr. down-time (s)
nginx	181.5	0.0202	0.2077	3.505	3.734	0.547
httpd	193.3	0.0807	0.196	3.19	3.467	1.712

We tend to vertically adjust the cloud infrastructures, thus we use coordinated vertical elasticity between the containers and their host machine. Using such mechanism, elasticity actions are coordinated and performed quickly.

3 ELASTIC CONTROLLER

3.1 General Design

Our system adheres to the control loop principles of the IBM's MAPE-K reference model (IBM, 2006). The control part of MAPE- K consists of many phases: Monitor, Analyze, Plan, and Execute. The managed components in this context are the infrastructure units KVM VMs and Docker containers, the containerized applications as well. We design elastic controllers to automatically adjust resources to the varying workload without violating QoS by growing or shrinking the amount of resources quickly on demand for both containers and their VMs. Figure 1 shows the general architecture of our controllers. The architecture design includes an elastic controller for Docker containers and another one for the hosting machine. The aim for the second controller is to allocate/de-allocate resources if containers residing on a virtual host machine require more/less resources than the amount of resources offered by that VM.

3.2 Components of the System

3.2.1 Monitoring Component

The monitoring component of Docker controller collects periodically current resource utilization and acquisition of every container on the host. The collected data can be *(i)* the resource utilization metrics such as CPU or memory current usage or *(ii)* the acquired resources such as memory size or CPU cores. This information is collected from the Docker daemon via its RESTful API and from the container cgroups filesystem directly. Our container controller monitors these cgroups each 4 seconds on an interval of 16 seconds, then the average values are reported. This monitoring data will be used in the reactive model in the elastic Docker controller to make elastic actions. Similarly, the host machine resource utilization and the amount of acquired resources are monitored periodically as shown in Table 2. The elasticity VM controller will use this data to provision/de-provision resources on the host machine. The values shown in Table 2 (thresholds, increase/decrease limits) are chosen following (Al-Dhuraibi et al., 2017a), (Dawoud et al., 2012), (Baresi et al., 2016) which are based on real-world best practices. We have noticed that the CPU and memory utilization values are sometimes fluctuating rapidly, which could be due to the nature of workload. Therefore, to avoid these oscillations, we measure CPU and memory utilization periodically on an interval of 16 seconds (as shown in Table 2), then we take the average values as the current utilization.

3.2.2 Docker Controller

Docker relies on cgroups to combine processes running in a container. Cgroups allow to manage the resources of a container such as CPU, memory, and network. Cgroups not only track and manage groups of processes but also expose metrics about CPU, memory, etc. Cgroups are exposed through pseudo-filesystems. In these files, Docker resources can be configured to have hard or soft limits. When soft limit is configured, the container can use all resources on the host machine. However, there are other parameters that can be controlled here such as *CPU shares* that determine a relative proportional weight that the container can access the CPU. Hard limits are set to

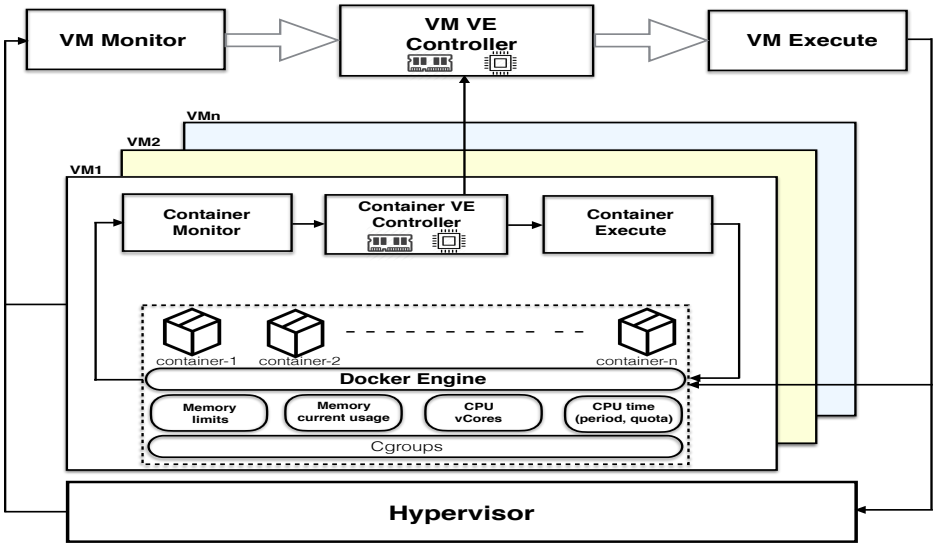

Figure 1: Coordinated elastic controllers between VMs and containers.

give the container a specified amount of resources, Docker vertical elasticity controller can change these hard limits dynamically according to the workload. The CPU access can be scheduled either by using Completely Fair Scheduler (CFS) or by using Real-Time Scheduler (RTS) (Red Hat, Inc.,). In CFS, CPU time is divided proportionately between Docker containers. On the other hand, RTS provides a way to specify hard limits on Docker containers or what is referred to as ceiling enforcement. Docker controller is integrated with RTS in order to make it elastic. When limits on Docker containers are set, the elasticity controller scales up or down resources according to demand. Once there is no limits set, it is hard to predict how much CPU time a Docker container will be allowed to utilize. In addition, as indicated, Docker can use all resources on the host machine by default, there is no control how much resources will be used, and the customer may not afford to pay the cost of such resources. The elastic controller of Docker containers adjusts memory and CPU vcores according to the application workload. This controller modifies directly the cgroup filesystems of the container to execute the elastic action (scaling up/down). The average CPU or memory usage values are calculated over fixed interval of time and compared against upper/lower thresholds (70% / 90%) as shown in Table 2.

When the thresholds are hit and the logical conditions are met, the controller increases or decreases the resources with values shown in Table 2. For example, if the average memory utilization for the last 16 seconds is greater than the upper threshold (90%), then increase the memory size by 256MB, and wait 10 seconds (breath up duration) before effectuating another scaling action.

Table 2: System control parameters.

Parameters	Docker containers	VMs
Upper threshold	90%	90%
Lower threshold	70%	70%
Period	4 sec	1 min
Interval	16 sec	1 min
Breath-up/down	10/20 sec	20/40 sec
CPU adaptation	1 vCPU	1 vCPU
Memory adaptation	-128/+256	-256/+512

In Table 2, we notice that memory adaptation values (increase and decrease ratios) are different. The controller decreases memory size by a small amount in the scaling down process because the applications are sensitive to the memory resource, and this could lead to interrupt the functionality of the application. In addition, after each scaling decision, the controller waits a specific period of time (breath duration). Breath duration is a period of time left to give the system a chance to reach a stable state after each scaling decision. As shown in Table 2, we set two breath durations, breath-up and breath-down. Breath-up/down is the time to wait after each scaling up/down decision, respectively. We choose small values for breath-up/down durations because the application adapts quickly to the container change . Breath-up is smaller than breath-down to allow the system to scale-up rapidly to cope with burst workload. Breath-down is larger than breath-up duration in order to avoid uncertain scaling down action. Our elastic Docker controller manages all the containers

residing on the virtual machine taking into consideration the available resources on that machine and the already allocated resources to the containers.

3.2.3 VM Controller

If containers allocate all resources on their hosting VM, they could reach an overload point of 100%. At that time the overload could cause errors in the workload execution since there is no free resources to provision. Therefore, our VM controller should intervene before such situation takes place. Likewise Docker containers, the hosting VM is monitored constantly and capacity is increased or decreased in relation to the VM reconfiguration policy involved in our VM controller. The VM controller performs vertical elasticity actions based on rules and real-time data captured by the monitoring system. As shown in Table 2, the monitoring component monitors the VM resource usage on an interval of one minute. It uses *psutil* library to get the resource metrics. The controller analyzes these collected data using its reactive model, it triggers its scaling decisions to increase or decrease VM resources, at the same time, it allows Docker engine to detect the new resources by updating cgroups of that Docker daemon. The values to increase/decrease memory, vCPUs are +512MB/-256MB, +1/-1, respectively.

3.3 Interactions Between Components

As shown in Figure 1, the VM controller can trigger elastic actions based on two cases: *(i)* when the VM resources utilization reaches certain thresholds, *(ii)* when it receives a demand from the Docker controller to increase or decrease resources. Here, the VM controller can increase resources without receiving a demand from the Docker controller if we suppose that there are other processes running on the VM alongside with containers. When the VM controller adds more vCPUs to the VM, the Docker engine does not detect these resources whether it uses hard or soft limits. Therefore, upon each scaling decision, the VM controller compares the resources on the VM and Docker engine, it then identifies the ids of the newly added vCPUs, then it updates the cgroups of Docker engine. Now, Docker engine can allocate these resources to containers. The coordination between the controllers is our major concern, we take the below scenario to illustrate a case of such coordination. Suppose that a VM has 3 vCPUs and three containers are deployed where hard limits are set and each container has 1 vCPU. If the first container usage is 100%, and the other two containers are idle (1 vCPU is 100%, 2 vCPUs are idle), the VM controller will try to decrease the vCPUs, but if it decreases the vCPUs, this will lead to destroy the container whose vCPU is withdrawn. Therefore, the coordination will prevent the VM controller to scale down, and the Docker controller will demand the VM controller to allocate more resources in order to give the first container more resources.

4 VALIDATION

4.1 Experimental Setup

We evaluated our work using RUBiS (OW2, 2008), a well-known Internet application that has been modeled after the internet auction website eBay. Our deployment of RUBiS uses two tiers: application tier, a scalable pool of JBoss application servers that run Java Servlets, and a MySQL database to store users and their transactions. We performed all our experiments on Scalair[2] infrastructure. Scalair is a private cloud provider company. We developed the experiments using the following technologies: (a) KVM version 1.5.3-105.el7_2.7 (x86_64), libvirt version 1.2.17, virt-manager 1.2.1, the number of VMs used and their characteristics will be described in the specific experiment subsections because we have used different configurations based on the objective of the experiment. (b) VMWare VCenter version 6.0. (c) Docker engine version 17.04.0-ce. (d) Kubernetes v1.5.2 (Brewer, 2015), the Kubernetes cluster consists of 3 machines. (e) ab (Apache HTTP server benchmarking tool) version 2.3 to generate workloads. The hardware specifications consist of 4 powerful servers: 2 HP ProLiant DL380 G7 and 2 HP Proliant XL170r Gen9. The experiments answer the following research questions (*RQ*):

- *RQ#1:* how can containers automatically use the hot added resources to their hosting VM?
- *RQ#2:* what is the efficiency of performing scaling decisions made by our coordinated controller?
- *RQ#3:* is our coordinated vertical elasticity of both VMs and containers better than vertical elasticity of VM only or vertical elasticity of containers only (i.e., $V_{cont}.V_{vm} > V_{vm} \oplus V_{cont}$)?
- *RQ#4:* is our coordinated vertical elasticity of both VMs and containers better than horizontal elasticity of containers (i.e., $V_{vm}.V_{cont} > H_{cont}$)?

[2]https://www.scalair.fr

4.2 Evaluation Results

We describe each experiment and analyze the results in response to the RQs.

RQ#1. In this experiment, we configure two VMs, each with Ubuntu Server 16.04.2 LTS. Initially, VM1 has 2 vCPUs with 2GB of RAM. We deploy RUBiS application inside two containers on VM1. The ab benchmark is installed on VM2, then we generate a workload to the RUBiS application (i.e., 600K requests, concurrency rate 200). The workload' requests query RUBiS database to retrieve lists of products, categories, items, etc of the auction website. The difference between workloads is the intensity and concurrency levels. We let the default policy for Docker containers which allow them to use all the available resources. The VM controller is enabled, we register the response time when the workload requests are finished, it was 588.846 seconds. In the second case, we run the same workload, however in this case, we enabled our coordinating controller and we set limits to Docker containers that will be reconfigured by the container controller to accommodate the charge and the response time was 487.4 seconds when the workload is finished. Based on these results, we conclude the following findings:

- The response time is high in the first case because Docker engine does not detect the added resources at the VM level. VM controller has added one vCPU to the VM (the total of CPUs moves to 3 on VM1), however, the two containers used only two CPUs, the third vCPU is idle because containers do not detect automatically the added resources.
- In the second case, the response time becomes smaller, thanks to our coordinated controller which allows containers to demand more resources and subsequently update the Docker engine with the added resources.
- The combined controller augments performance by **20.8%** in this experiment. However if the workload increases, the coordinated controller will accommodate resources in contrary to the first case where the containers can not use more that the initially allocated 2 vCPUs.

RQ#2. In this evaluation, we measure the execution time of elastic actions. Elastic action is the process of adding or removing resources (CPU or memory) to a container, a KVM VM, or a VMware VM. We repeat the experiment eleven times for each resource (CPU or memory) on each target (i.e., container, KVM VM, VMware VM), and each time the action consists of 15 scaling up or down actions. During the experiments, the resources experience different stress workloads. We execute elastic actions and we measure the time they take to resize the resource, and then the median and variance is calculated. We take these measures to illustrate the efficiency of our approach to execute auto-scaling actions and to show the differences between the different virtualization units and technologies. We compute the average execution time, median time, and variance for Containers, KVM and VMware VM respetcively: (0.010s,0.009s, 0.000004), (3.29, 3.02s, 2.97) and (47.58, 44.14s, 45.44).

Based on the these values, we conclude:

- The average execution time is close to median time which indicates that the execution of the elastic actions are stable.
- The elastic actions performed in containers are faster than resizing KVM VM or VMware VM. There is no comparison between containers adaptation and hypervisors, the containers adapt more quickly to the reconfiguration changes while it takes more time to execute scaling actions against hypervisors. The VMware hypervisor manged by VCenter takes more time. High workloads lead to slow execution of elastic actions, particularly in VMware, i.e., why the variance is high.

RQ#3. This experiment provides a comparison among vertical elasticity of containers (V_{cont}), vertical elasticity of VMs (V_{vm}), and our proposed approach, coordinating elasticity of both containers and VM ($V_{vm}.V_{cont}$), in terms of performance, i.e., the execution time of workloads and mean response time of concurrent requests. We run three scenarios in this experiment, each scenario has its specific configuration. Five workloads drive each scenario. The experiment runs on 4 VMs: VM1, VM2, VM3, and VM4. VM4 used to generate the ab benchmark. In the first scenario ($scenario_1$), RUBiS application is deployed on two containers in VM1 which has 3vCPUs, 2GB of RAM, initially, each Docker container has 1vCPU and 512MB of RAM. We enabled the elastic container controller (which will allow to use the resources available on the hosting VM) and it is named ElasticDocker controller. We measure the total execution time and the mean response time of concurrent requests for each workload as shown in Figure 2. In $scenario_2$, we deploy RUBiS application on one VM (VM2) and its database in another VM (VM3). The VMs have 1 vCPU and 1GB of RAM each. We enabled the vertical VM controller to adjust resources according to workload demand and then register the total execution time and the mean response time of concurrent requests for each workload as shown in Figure 2. In $scenario_3$, we use the same configuration as in $scenario_1$, except that we enabled our coordinating controller which controls elasticity of con-

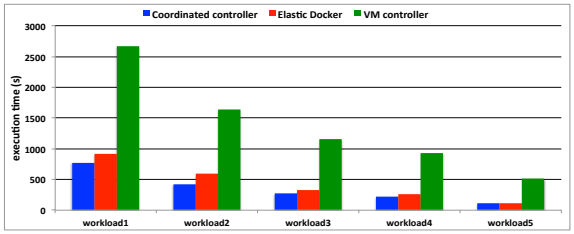

Figure 2: Workloads execution time.

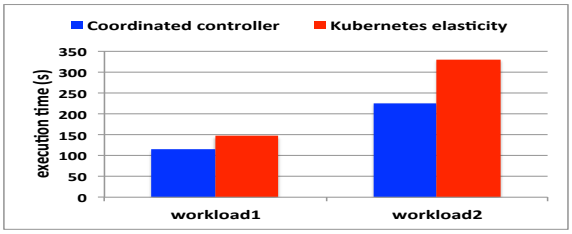

Figure 3: workloads total execution time.

tainers on the VM, and if there are no enough resources, it will add resources to the VM level. In Figure 2, the red color represents scenario$_1$, the green color represents scenario$_2$ and the blue color represents scenario$_3$. Based on the analysis of this experiment, we concluded the following findings:

- In scenario$_1$, the average total execution time, and the mean response time across concurrent requests for the five workloads is **443,7** seconds and **0,91** ms, respectively. Similarly, the average total execution time for the five workloads in scenario$_2$ and scenario$_3$ is **1383,4** seconds and **362,1** seconds, and the mean response time across concurrent requests is **3,1** ms and **0,76** ms, respectively.
- The combined vertical elasticity (scenario$_3$) outperforms the container vertical elasticity (scenario$_1$) by **18.34%** and the VM vertical elasticity (scenario$_2$) by more than **70%**. However, if more workloads are being added to the scenario$_1$, it will not handle them because the available resources will be consumed and performance will be degraded. This demonstrates that the equation $V_{cont}.V_{vm} > V_{vm} \oplus V_{cont}$ is true.

RQ#4. The aim of this experiment is to provide a comparison between horizontal elasticity of containers and our coordinating vertical elasticity of VMs and containers. We use Kubernetes horizontal elasticity. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. To achieve the experiment, we use Kubernetes version v1.5.2. Our deployment of RUBiS on Kubernetes uses three tiers: a load-balancer (we use Kubernetes service to perform this role), a scalable pool of JBoss application servers, and a MySQL database. Kubernetes platform is deployed on 3 nodes running CentOS Linux 7.2. RUBiS is deployed in two containers, in addition to a load balancer. Then, we set the Kubernetes Horizontal Pod Autoscaling (HPA) to scale RUBiS containers based on rule-based thresholds. We use the same thresholds used in scenario$_3$ in the previous section (in RQ#3). We generate two workloads to both our coordinated controller and Kubernetes cluster. The total execution time across all concurrent requests are measured for each workload as shown in Figure 3. According to these results, we conclude the following findings:

- The total execution time for the workloads is **340,66** seconds when our elastic controller is used, while it is **475,558** seconds when Kubernetes HPA is used. The execution time is longer when Kubernetes is used due to the slow Kubernetes integrated monitoring system (Heapster).
- Our combined vertical elasticity outperforms the horizontal elasticity by **39.6%** according to the results of this experiment.
- This proves the equation $V_{vm}.V_{cont} > H_{cont}$ is true.

5 RELATED WORK

We present works related to elasticity particularly the vertical elasticity of both VMs and containers. For the VM vertical elasticity, there are some works which focus on CPU resizing, e.g, (Lakew et al., 2014) and (Dawoud et al., 2012), while others concentrate on memory resizing, e.g., (Baruchi and Midorikawa, 2011) as well as combination of both such as the work of (Farokhi et al., 2015). (Monsalve et al., 2015) proposed an approach that controls CPU shares of a container, this approach uses CFS scheduling mode. Nowadays, Docker can use all the CPU shares if there is no concurrency by other containers. (Paraiso et al., 2016) proposed a tool to ensure the deployability and the management of Docker containers. It allows synchronization between the designed containers and those deployed. In addition, it allows to manually decrease and increase the size of container resources. (Baresi et al., 2016) proposed horizontal and vertical autoscaling technique based on a discrete-time feedback controller for VMs and containers. This novel framework allows resizing the container in high capacity VM, however, it does not control VM in response to container workload. It triggers containers to scale out horizontally to cope with workload demand. In addition, the application requirements and metadata must be precisely defined to enable the system to work. It also adds overhead by inserting agents for each container and VM. (Al-Dhuraibi et al., 2017a) describe an approach that manages container vertical elasticity, and when there is no

more resources on the host, they invoke live migration. Kubernetes and Docker Swarm are orchestration tools that permit container horizontal elasticity. They allow also to set limit on containers during their initial creation. The related works either trigger horizontal elasticity or migration to another high capacity machines. Our proposed approach supports automatic vertical elasticity of both containers and VMs, at the same time, container controller invokes VM controller to trigger scaling actions if there is no more resources on the hosting machine. Our work is the first one that explores the coordination between vertical elasticity of containers and VMs.

6 CONCLUSION

This paper proposes a novel coordinated vertical elasticity controller for both VMs and containers. It allows fine-grained adaptation and coordination of resources for both containers and their hosting VMs. Experiments demonstrate that: (i) our coordinated vertical elasticity is better than the vertical elasticity of VMs by 70% or the vertical elasticity of containers by 18.34%, (ii) our combined vertical elasticity of VMs and containers is better than the horizontal elasticity of containers by 39.6%. In addition, the controller performs elastic actions efficiently. We plan to experiment this approach with different classes of applications such as RTMP to verify if same results will be obtained with the predefined thresholds. Our future work also comprises the integration of a proactive approach to anticipate future workloads and reacts in advance. Furthermore, we plan to address hybrid elasticity or what we called diagonal elasticity: integrating both horizontal and vertical elasticity.

ACKNOWLEDGEMENTS

This work is supported by Scalair company (scalair.fr) and OCCIware (www.occiware.org) research project.

REFERENCES

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2017a). Autonomic Vertical Elasticity of Docker Containers with ElasticDocker. In *10th IEEE International Conference on Cloud Computing (CLOUD)*, Hawaii, US. To appear.

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2017b). Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing*, PP(99):1–1.

Appuswamy, R., Gkantsidis, C., Narayanan, D., Hodson, O., and Rowstron, A. (2013). Scale-up vs Scale-out for Hadoop: Time to Rethink? In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 20:1–20:13, New York, NY, USA. ACM.

Baresi, L., Guinea, S., Leva, A., and Quattrocchi, G. (2016). A Discrete-time Feedback Controller for Containerized Cloud Applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016*, pages 217–228, New York, NY, USA. ACM.

Baruchi, A. and Midorikawa, E. T. (2011). A Survey Analysis of Memory Elasticity Techniques. In *Proc. of the 2010 Conf. on Parallel Processing*, Euro-Par 2010, pages 681–688, Berlin, Heidelberg. Springer-Verlag.

Brewer, E. A. (2015). Kubernetes and the Path to Cloud Native. In *Proc. of the Sixth ACM Sym. on Cloud Comp.*, SoCC '15, pages 167–167, New York, USA. ACM.

Cecchet, E., Marguerite, J., and Zwaenepoel, W. (2002). Performance and Scalability of EJB Applications. *SIGPLAN Not.*, 37(11):246–261.

Checkpoint/Restore (2017). Website https://criu.org/Checkpoint/Restore.

Coutinho, E. F., de Carvalho Sousa, F. R., Rego, P. A. L., Gomes, D. G., and de Souza, J. N. (2015). Elasticity in Cloud Computing: a Survey. *Annals of Telecommunications*, pages 1–21.

Dawoud, W., Takouna, I., and Meinel, C. (2012). Elastic virtual machine for fine-grained cloud resource provisioning. In *Global Trends in Computing and Communication Systems*, pages 11–25. Springer.

Farokhi, S., Lakew, E. B., Klein, C., Brandic, I., and Elmroth, E. (2015). Coordinating CPU and Memory Elasticity Controllers to Meet Service Response Time Constraints. In *2015 International Conf. on Cloud and Autonomic Computing (ICCAC)*, pages 69–80.

IBM (2006). An Architectural Blueprint for Autonomic Computing. *IBM White Paper*.

Lakew, E. B., Klein, C., Hernandez-Rodriguez, F., and Elmroth, E. (2014). Towards Faster Response Time Models for Vertical Elasticity. In *Proceedings of the 2014 IEEE/ACM 7th Int. Conf. on Utility and Cloud Computing*, UCC '14, pages 560–565, Washington, USA.

Monsalve, J., Landwehr, A., and Taufer, M. (2015). Dynamic CPU Resource Allocation in Containerized Cloud Environments. In *2015 IEEE International Conference on Cluster Computing*, pages 535–536.

OW2 (2008). *RUBiS: Rice University Bidding System*. http://rubis.ow2.org [Accessed: Whenever].

Pahl, C. (2015). Containerization and the PaaS Cloud. *IEEE Cloud Computing*, 2(3):24–31.

Paraiso, F., Challita, S., Al-Dhuraibi, Y., and Merle, P. (2016). Model-Driven Management of Docker Containers. In *9th IEEE Int. Conf. on Cloud Computing (CLOUD)*, pages 718–725, San Franc., United States.

Red Hat, Inc. Managing system resources on Red Hat Enterprise Linux. .

Sztrik, J. (2012). Basic queueing theory. *Univ. of Debrecen, Fac. of Informatics*, 193.