International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Auto-Scaling of Micro-Services Using Containerization

Priyanka P. Kukade[1], Prof. Geetanjali Kale[2]

[1, 2] Pune Institute of Computer Technology, Dhankawadi, Pune - 411 043, India

Abstract: ***Cloud computing has emerged as a new computing model, where storage, network and computation is provided as a service to user, who can lease and release the service on demand. To enable the user of cloud environment to develop and deliver solutions, the cloud provider has to build a robust set of services to support the customers. These are Platform services. The cloud service provider has to adhere to strict customer (Service Level Agreement) SLAs. To support these SLAs, platform services need to be built with certain principles. We present an approach of restructuring platform services into micro-services. These micro-services are nothing but loosely coupled and independently deployable services. Deployment of service in cloud needs to be dynamic. You need to deal with failure cases and provide high availability. It is required to manage more instances of application when the demand increases and to scale down for fewer requests for conserving energy. The traditional use of virtual machine to deploy services lead to low performance. OS level virtualization approach is adopted to deploy services. Here we have proposed an approach for elastic scaling of services in cloud environment.***

Keywords: Cloud Computing, Virtualization, PaaS, SLA.

1. Introduction

Cloud computing is an emerging computing model. It is an on demand service model, which provides infrastructure, platform and software as a service. That means software or platform will be available in the form of service. Platform is nothing but a set of services. Platform services provide a platform and environment to allow developers to build applications and services over the internet [9] [10].

Today more applications are being deployed cloud environment. Just deployment of service in cloud is static. You need to deal with failure cases and provide high availability. If a container fails or service within the container fails there should be mechanism to deal with it. Also elasticity is the key concept in cloud, which allows dynamically allocating and freeing resources as per request [11] [12] [13]. There is need of scaling up and scaling down the services requested by the customer. Thus there is need of orchestration of container.

In the proposed system, micro-services approach is used to overcome the draw-back of monolithic architecture. Use of containers for the deployment of services in the proposed system, instead of traditional use of virtual machines leads to improvement in performance. An auto-scaling approach is proposed which would scale the services based on request and memory load.

2. Literature Survey

Elasticity property of cloud is gaining more attraction of people. Horizontal and vertical scaling are the ways by which scaling can be accomplished. Horizontal scaling deals with adding or removing of resources that are of the same type. Vertical scaling deals with replacing the existing resource with the lower or higher capacity. Different approaches have been followed by researcher, enhancing the scalability. Here are the few strategies of application scaling in cloud.

Application scalability issues and strategies in different Service model of cloud are explained by Vaquero, Luis M., Luis Rodero-Merino, and Rajkumar Buyya [1]. Scaling in Infrastructure as a service (IaaS) can be done in two ways - horizontal scaling and vertical scaling. Here author has classified two ways of achieving scalability in Platform as a service (PaaS) model- the container and the database management system (DBMS) scalability. User component is deployed and run in the software platform called container. Different platforms can be used by different PaaS providers. Author suggested that for the components hosted by PaaS, platform type can define different lifecycles, services and APIs. On the other hand, databases provide data persistence support. It should address the demand for data transactions support combined with big availability and scalability requirements.

The auto-scaling problem for a more general application model is explored and allows individual (non-uniform) job deadlines [2]. Here the aim is to complete all jobs by using resources that are less costly within the given deadline. Deadlines are used as a performance requirements specified by the users, and deadline misses are not strictly prohibited. Web request response time, network latency and a program's running time is selected as a deadline. Deadline assignment techniques are used to calculate an optimized resource plan for each job and the number of instances using the Load Vector idea is determined. Job scheduling and resource scaling is addressed at the same time by considering both the job-level and global-level cost-efficiency.

A lightweight approach is proposed by Han, Rui, et al for achieving more cost efficient scaling of cloud resources at the IaaS cloud provider's side is proposed [3]. The multi-tiered applications, which are already implemented using multiple VM's have improved resource utilization between VM's as application demands vary. It proposed the following: Fine-

grained scaling approach, Improving resource utilization, Implementation and experimental evaluation. They have proposed an intelligent platform based on the LS (light weight scaling) algorithm, which is implemented to automate the scaling process of cloud applications.

Automatic scaling problem is summarized in the cloud environment by Sharma, Tejinder, and Vijay Kumar Banga [4]. It is modeled as a modified Class Constrained Bin Packing (CCBP) problem. Here each server is a bin and each class represents an application. Here an innovative auto scaling algorithm to solve the problem and present a rigorous analysis on the quality of it with provable bounds is made. Compared to the existing Bin Packing solutions, item departure is supported which can effectively avoid the frequent placement changes caused by repacking. It support green computing by adjusting the placement of application instances adaptively and putting idle machines into the standby mode.

To overcome the general lack of effective techniques for workload forecasting and optimal resource allocation Roy et al [5], made contributions. These contributions were - discusses the challenges involved in auto scaling in the cloud. Develope a model-predictive algorithm for workload forecasting that is used for resource auto scaling. Finally, their algorithm provided empirical results that demonstrate that resources can be allocated and deallocated in a way that satisfies both the application QoS while keeping operational costs low.

An auto-scaling system, WebScale, was developed by Ching et al [6], which is not subject to the aforementioned constraints, for managing resources for Web applications in data centers. They devised a new method for analyzing the trend of workload changes after comparing the efficiency of different scaling algorithms for Web applications. The experiment results showed that even when facing sudden load changing WebScale can keep the response time of web applications low.

Feasibility simulation of using virtualization technology to auto-scaling problem in cloud computing was proposed by Thepparat et al [7]. It built two different models using ARENA simulation software. There are auto-scaling without server virtualization and auto-scaling with server virtualization. The results demonstrated that life time of servers and CPU utilization increases by employing virtualization technology.

3. Proposed System

3.1 Micro-services

Micro services are a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services [8]. The micro service architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

In micro services pattern, a software system is comprised of a number of independently deployable services, each with a limited scope of responsibility. Less frequently, micro-services may rely on lower level services. Here each micro-service can be developed, managed and scaled independently throughout its life-cycle. A well implemented micro services architecture also ensures that the overall system is able to gracefully degrade its functionality when one or more services are offline.

3.2 Containerization

It is also known as Operating System Virtualization. As the name implies, the abstraction is the operating system itself, instead of the platform. It is an approach to virtualization in which the virtualization layer runs as an application within the operating system (OS). Here, the operating systems kernel runs on the hardware with several isolated guest virtual machines (VMs) installed above it. The isolated guest virtual machines are called containers.

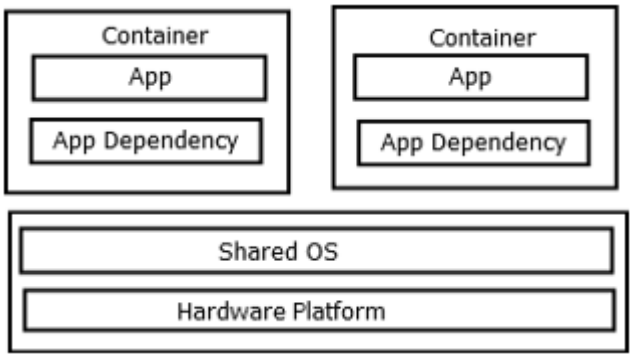

Figure 1: Containerization

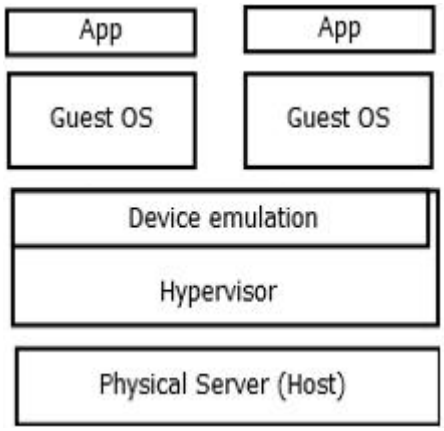

Figure 2: Traditional hardware virtualization

Here the operating system provides a set of user-spaces that are isolated from one another, but offers the abstraction necessary such that applications believe that they are part of the singular user-space on the host. With container-based virtualization, the overhead associated with having each guest run a completely installed operating system is not there. Here there is just one operating system taking care of hardware calls, thus it improves performance. However each guest must use the same operating system the host uses, which results in restriction for user.

4. Auto Scaling Approach

Figure 3 shows architecture of the system. It is master slave architecture. Here nodes are nothing but slave, where the service containers are deployed. Master receives request for the services and routes it to the running container. The auto scaling module is added in master. It includes the following module:

- **Service Container Monitor**
  It is responsible to monitor the corresponding container for each service. It also monitors the service URL for these containers.
- **Request Monitor**
  It is responsible for monitoring the request count for each container of the service. If the request count for a container exceeds threshold, new replica of container are started. If the request count of container is below threshold then container are scaled downed and a minimum of one replica are maintained.
- **Memory Load Monitor**
  It is responsible for monitoring the memory load for each container of the service. If the memory load for a container exceeds threshold, new replica of container are started. If the memory load of container is below threshold then container are scaled downed and a minimum of one replica are maintained.
- **Scale**
  This module is responsible for scaling the container up and down. Here new containers are started and stopped as per the request from memory load monitor and request monitor.

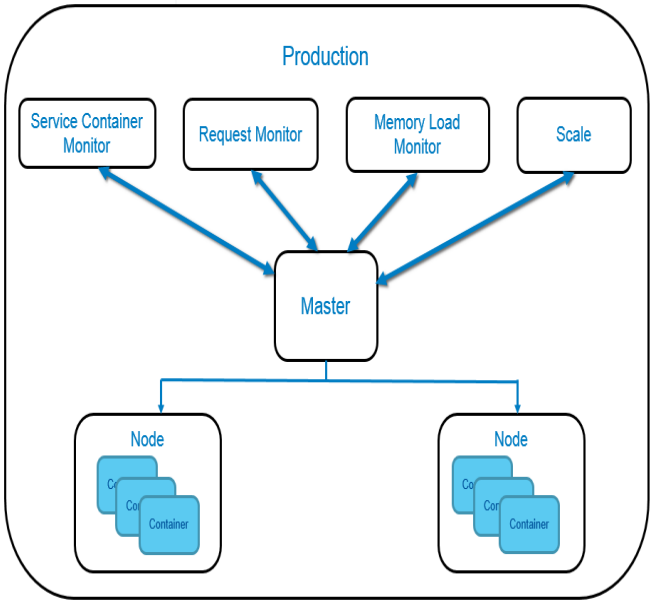

Figure 3: System architecture

4.1 Data tables

For the scaling of the services a request monitor is used, which monitors new request and updates the Service request data table as shown in table 1. It contains the information related to service name, Request Count elapsed time and request per minute (rpm).

Table 1: Service request data

Service Name	*Request Count*	*Elapsed time*	*RPM*
Audit	248	23	10
Keyword	5	0	5
Rabbitmq	1	0	1
Update	23	21	1

A memory monitor is used in auto-scaler module, which monitors memory load of each service and updates the Memory load data table as shown in table 2. It contains the information related to service name, No of Container Instances and Memory load for each container.

Table 2: Memory load data

Service Name	*No. of Container*	*Memory load*
Audit	1	62.0058
Keyword	1	64.6663
Rabbitmq	2	67.0605 78.4295
Update	1	69.5956

For scaling services, service container information is maintained in Service container data table as shown in table 3 and is updated when the container instances increases or decreases. It contains the information related to service name, Replication Controller (RC), Replica, Max threshold (THmax) and Min threshold (THmin) for each service.

Table 3: Service Container data

Service Name	RC	Replica	THmax	THmin
Rabbitmq	Rc-rabbitmq	2	10	4
Audit	Rc-audit	1	15	7
Keyword	Rc-keyword	1	11	4
Update	Rc-update	1	13	6

5. Results

The system is analyzed for auto-scaling approach by increasing and decreasing the HTTP request for one of the service. Request rate for the service is taken to be 60 requests per minute. Figure 4 represents the scale up of container instances of service when the request load increases.

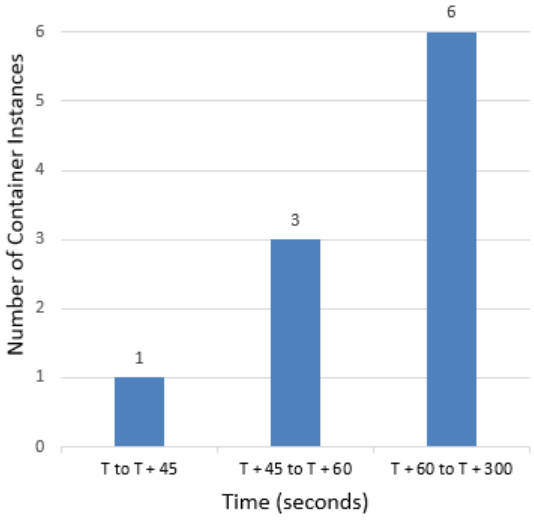

Figure 4: Scale up

Figure 5 represents the scale down of container instances of service when the no more requests is send to the service.

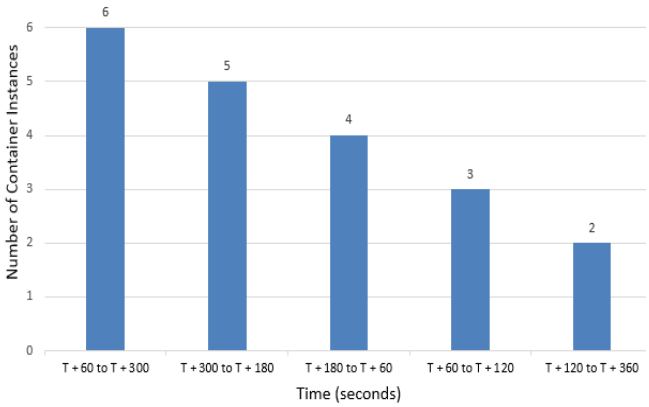

Figure 5: Scale down

6. Conclusion

Today more applications are being deployed in cloud environment. To overcome the problems of monolithic application we are using micro-services approach. Virtual machines are used traditionally to achieve isolation and resource control. The use of virtual machines for deploying application results in low performance and scalability. Thus here we are using containerization instead of traditional virtualization.

Deployment of service using cloud is static. You need to deal with failure cases and provide high availability. If a container fails or service within the container fails there should be mechanism to deal with it. So just deployment of service in container is not enough. It is required to manage more instances of application when the demand increases and to scale down for less request for conserving energy. Thus use orchestration tool is required for managing the health and elastic scaling of micro-services.

We have presented the design of a system used for developing and automatically deploying micro-services on cloud infrastructure. Through the literature survey we have identified and analyzed the existing solution for deployment of services in cloud. We proposed an approach to elastic scaling of micro-services in cloud.

References

[1] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, “Dynamically scaling applications in the cloud,” ACM SIGCOMM Computer Communication Review, vol. 41, pp. 45–52, April 2011.
[2] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” in International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 49–55, June 2011.
[3] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, “Lightweight resource scaling for cloud applications,” in International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 644–651, Sep. 2012.
[4] Z. Xiao, Q. Chen, and H. Luo, “Automatic scaling of internet applications for cloud computing services,” Computers, IEEE Transactions on, vol. 63, pp. 1111–1123, July 2014.
[5] N. Roy, A. Dubey, and A. Gokhale, “Efficient autoscaling in the cloud using predictive models for workload forecasting,” in IEEE International Conference on Cloud Computing (CLOUD), pp. 500–507, July 2011.
[6] C.-C. Lin, J.-J. Wu, J.-A. Lin, L.-C. Song, and P. Liu, “Automatic resource scaling based on application service requirements,” in IEEE 5th International Conference on Cloud Computing (CLOUD), pp. 941–942, Aug. 2012.
[7] T. Thepparat, A. Harnprasarnkit, D. Thippayawong, V. Boonjing, and P. Chanvarasuth, “A virtualization approach to auto-scaling problem,” in Eighth International Conference on Information Technology: New Generations, pp. 169–173, June 2011.
[8] M. fowler, “Micro-services.” http://martinfowler.com/articles/microservices.html, 2015. [Online; accessed 2-July-2015].
[9] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” Journal of internet services and applications, vol. 1, pp. 7–18, Dec. 2010.
[10] Y. Zhang, Y. Li, and W. Zheng, “Automatic software deployment using userlevel virtualization for cloud-computing,” Future Generation Computer Systems, vol. 29, pp. 323–329, Jan. 2013.
[11] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” technology, vol. 28, pp. 32–36, Feb. 2014.
[12] R. Dua, D. Kakadia, et al., “Virtualization vs containerization to support paas,” in International Conference on Cloud Engineering (IC2E), pp. 610–614, Aug. 2014.
[13] M. J. Scheepers, “Virtualization and containerization of application infrastructure:A comparison,” 21st Twente Student Conference on IT ,pp. 1–7, Feb. 2014.