Elastic VM for Rapid and Optimum Virtualized Resources' Allocation

Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel
Hasso Plattner Institute
Potsdam University
Potsdam, Germany
E-mail: firstname.lastname@hpi.uni-potsdam.de

***Abstract*—The rapid growth of E-Business and the frequent changes in sites contents pose the need for rapid and dynamic scaling of resources. Cloud computing infrastructure, based on virtualization technologies, enables agile and dynamic scalability of resources. However, current implementation of the scalability in the cloud uses the Virtual Machine (coarse-grained) as a scaling unit which often leads to over-provisioning of resources. Hence, we propose Elastic VM (fine-grained) scaling architecture. It implements the scalability into the VM resources level to optimize resources' allocation. We support our approach by analytical analysis and describe in details how Elastic VM architecture improves the performance and optimize the dynamic allocation of resources.**

I. Introduction

To maintain a good quality of service (QoS), system administrators should provision adequate resources to cope with workload demand fluctuations. Unfortunately, over-provisioning implies extra cost and decreases the business profit. On the other hand, under-provisioning degrades the QoS, irritates the clients, and consequently retreats the E-Business durability.

Traditionally, hardware provisioning is implemented as steps done monthly or yearly according to the enterprise's policy and the expected workload. As seen in figure 1, in most cases, hardware upgrade implies over-provisioning (i.e., money loss) while new hardware is typically planned to run for the next months or years. However, at some point of time, the current hardware will be unable to cope with actual demand which results in QoS degradation.

The later advance in virtualization technology software, e.g. Xen [2] and VMware [3], enables cloud computing environment to deliver agile, scalable, and low cost infrastructures. However, as seen in figure 1, there is a gap between the actual demand and the automated resources allocation implemented by current scalability architectures in the clouds. From the client side this gap implies additional price. From the provider side it implies running additional hardware, consuming more power, and contributing more in CO_2 emissions [4].

Our intention in this paper is to reduce the gap between resources allocation and the actual demand. For this goal, we analyze the current scalability implementation offered by cloud infrastructure providers, e.g. Amazon EC2 [5] and GoGrid [6].

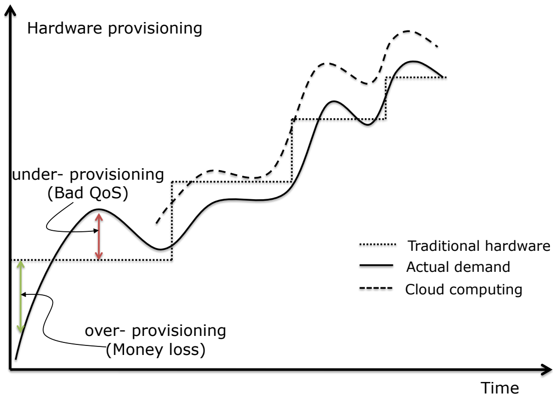


Fig. 1: Automated elasticity [1]

The analysis shows the following characteristics: First, the current scalability implementation uses VM as a scaling unit (i.e., coarse-grained scaling). Second, it implies running additional load-balancer VM instances.

In this paper, we propose Elastic VM scaling architecture; it is a vertical scaling architecture that scales the VM resources dynamically to cope with workload demand. To enable the dynamic scalability, the Elastic VM kernel is modified to accept on-the-fly resources scaling without interrupting the service or rebooting the system. Moreover, the hypervisor is extended with interfaces that enable modifying VMs resources by programming languages.

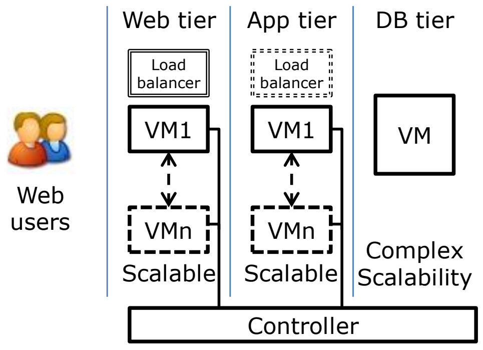
The rest of this paper is organized as follows: The next section overviews the common web applications patterns and scalability architectures. Section III compares the performance of both Multi-instances and Elastic VM architectures analytically. Section IV investigates pros and cons of both scalability architectures' implementation. Section V discuss the related work. Finally, section VI concludes the paper and points out to our immediate future work.

II. Scalability Architectures

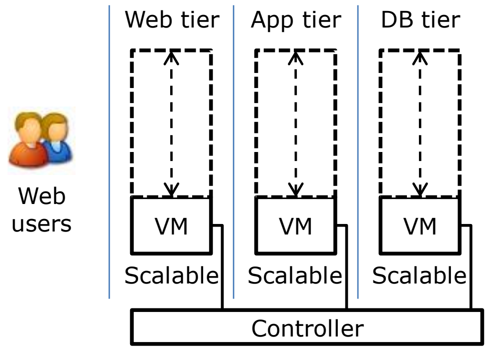
Typically, web applications consist of three tiers: web tier, application tier, and database tier. The incoming requests go through these tiers to get back to the user with the results. According to application characteristics, each tier may cast an intensive demand to specific resources making it a bottleneck. The dependency between the tiers propagates performance degradation of one tier to the whole application. Therefore, to cope with traffic demand and avoid QoS degradation, the first step is to predict the bottleneck tier, and then to scale it dynamically.

A. *Multi-instances Scaling Architecture*

Current implementation of the scalability by Amazon EC2 and GoGrid is abstracted in figure 2(a). Users' requests are directed to a load-balancer which forwards them to available VM instances (i.e., VM1 to VMn). To maintain a determined QoS, a controller monitors performance metrics (e.g., CPU utilization) and scale-out VMs to cope with workload demand surge or scale-down VMs to reduce the cost.



(a) Multi-instances Scaling Architecture



(b) Elastic VM Scaling Architecture

Fig. 2: Multi-tier Systems Scalability Implementation

B. *Elastic VM Scaling Architecture*

Elastic VM is a VM that supports scaling resources on-the-fly without interrupting the service or rebooting the system. To cope with workload demand, as shown in figure 2(b), a controller monitors tiers performance. If the monitored performance metric (e.g., CPU utilization) exceed a specified threshold, the controller scale up the virtual machine resources (e.g., CPU allocation) to maintain an acceptable system performance.

III. Performance Analysis

In this section we will analyze analytically the average response time of two architectures: 1-Multi-instances architecture running many instances each with one virtual CPU (vCPU). 2-Elastic VM which is scaled to run many vCPUs. The response time is calculated as the sum of the service time and the waiting time spent by the packet in the queue until being served. As seen in figure 2(a), each web-server could be modeled as a single-server queue model (M/M/1), on the other hand, Elastic VM in figure 2(b) could be modeled as a single queue with multiple servers (M/M/c).

According to Kendall's notation, M is a notation for Markovian (exponential) distribution, which means that both the inter-arrival time and service time are exponentially distributed. However, in case of Multi-instances, each queue is served by one server (i.e., one vCPU), while in case of Elastic VM, a single queue is served by c servers (i.e., vCPU). Equations that describe M/M/c model are as the following: System utilization ρ is calculated by:

$$\rho = \frac{\lambda}{c * \mu} \tag{1}$$

The probability of having zero requests:

$$P_0 = \left[\sum_{n=0}^{c-1} \frac{\lambda^n}{n!} + \frac{\lambda^c}{c!(1-\lambda/c)} \right]^{-1} \tag{2}$$

Expected average queue length:

$$E(m) = P_0 \frac{\rho^{c+1}}{c.c!} \frac{1}{(1-\rho/c)^2} \tag{3}$$

Expected average number of requests in the system:

$$E(n) = E(m) + \rho \tag{4}$$

Expected average total time in the system:

$$E(v) = E(n)/\lambda \tag{5}$$

Expected average waiting time in the queue:

$$E(w) = E(v) - 1/\mu \tag{6}$$

An example to compare the average response time of Multi-instances with Elastic VM:

Firstly, consider a Multi-instances architecture running 4 VMs instances in parallel, each machine is modeled as single queue served by one server (vCPU). Assuming that each vCPU has the capacity to serve 100 req/sec and the total traffic rate to the whole system is 320 req/sec. The incoming traffic is distributed fairly by the load balancer to be 80 req/sec for each VM instance. In this case the VM utilization $\rho = \frac{80}{100} = 80\%$. Hence, *the average response time* = 0.05 and the *waiting time* = 0.04 calculated by equations 5 and 6 in consequence where c=1, λ =80, and μ =100.

Secondly, consider the same traffic directed to Elastic VM with 4 vCPUs (i.e., c=4), in this case, the system utilization is calculated as $\rho = \frac{320}{4*100} = 80\%$, while *the average response time* = 0.017 and *the average waiting time* = 0.007 calculated by equations 5 and 6 in consequence where c=4, λ =320, and μ =100.

Substituting any value of c $>$ 1 in equations 1 to 6 proves that a single VM with multiple cores, as in Elastic VM implementation, performs better than many VMs instances each with one core running in parallel, as in Multi-instances architecture, even though there is the same total number of vCPUs in both systems.

IV. Elastic VM vs. Multi-Instances Architecture Implementation

By analyzing the scaling architectures implementations characteristics of both Elastic VM and Multi-Instances architecture, we can summarize pros and cons of each architecture as follow:

TABLE I: Comparison between Multi-instances and Elastic VM architecture implementation

Multi-instances architecture	Elastic VM architecture
Implies running load-balancer (i.e., additional consumption of resources)	No need for running additional VM instance as a Load-balancer
Limited to specific applications	Applicable to any tier
Applies VM as a scaling unit (coarse-grained scale)[1]	Fine-grained scaling while it implements scaling into VM resources
Scaling-down can interrupt sessions based web connections	Supports sessions based web connections
Booting time of VMs, to scale-out, increases the overhead	Eliminates the overhead caused by booting VMs
Scale-out overhead causes SLO violation and decreases the throughput	Scale-up vertically reduces SLO's violation and maintains higher throughput
Both software and hardware load-balancer can be a single point of failure	Elastic VM itself could be a single point of failure [2]
It supports business at all scales: small, medium, and big	Elastic VM is limited to one physical host, which limits it to small and medium scale business [3]

[1] One solution is to have smaller VM instances (e.g., Amazon micro-instance). However, this could reduce the probability of over-provisioning but it does not eliminate it totally.

[2] Compared to static machines, Elastic VM's ability to scale-up makes it more resistance to failure that could be caused by overloading. This characteristic makes it a recommended replacement to static load-balancer instances as an example.

[3] If a global policy is enabled to relocate VMs according to host utilization, it will be possible to move VMs with lower load into another hosts. This will increase the chance of the Elastic VM to scale-up dynamically without interrupting the service.

One of the challenging issues in Elastic VM is that some applications could be unaware of on-the-fly scaling, especially memory scaling. Fortunately, many research are directed to optimizing application parameters according to available resources like [7],[8],[9], and [10]. Such approaches can be integrated to our architecture to tune applications parameters for optimum performance after each scaling.

V. Related Work

Towards avoiding bottleneck tiers, Iqbal et al. [11] implemented a prototype using Multi-instances scaling architecture. The approach considers scaling database layer horizontally, but did not discuss associated challenges (e.g., data replication and synchronization) which could affect the approach feasibility and performance.

Many researchers presented analytical models to describe different tiers behavior, for example, Bhuvan Urgaonkar [12] presented multi-tier model based on a network of queues, while each queue represents a different tier. The scalability of this model is implemented by dispatching new instances at each tier except database tier which is not replicable in this model.

Using regression analysis of CPU utilization and service time to predict the bottlenecks, Dubey et al. [13] demonstrated an approach for performance modeling of two-tier applications (web and database). Even that the approach does not imply dynamic scaling, but it helps understanding application behavior for optimum capacity planning.

Using queuing theory models along with optimization techniques, Jung et al. [14] presented off-line techniques to predict system behavior and automatically generate optimal system configurations. The result is a rule set that can be inspected by human system administrators and used directly with a rule-based system management engines.

Amazon EC2 Spot Instances [5] is a way to provide VMs with a lower price. It is developed to serve customers who are in need for high computing but for none online systems (e.g., Video rendering; Scientific research; and Financial modeling and analysis). In fact, Amazon EC2 Spot Instances is one of the motivating ideas to our research. Nowadays, Amazon static *Large EC2* instance costs \$0.34 per hour, while static *Extra Large EC2* instance costs \$0.68 per hour. Implementing Elastic VM can emerge the following service: *Elastic Large to Extra Large EC2* instance which costs for example \$0.40 per hour. The lower case is to have a Large EC2 instance, and the upper case is to expand it to Extra Large EC2. As in Amazon EC2 Spot Instances, the idea behind the cost reduction is the dependency on the free capacity in cloud provider. Such approach depends on the probability of having free resources in the same zone, which is not guaranteed. However, having a global plan to manage the capacity and run a complementary workload on the same zone, considering the daylight differences around the world, increases the probability of having free resources in the same host.

VI. Conclusion & Future Work

In this paper, we propose Elastic VM scalability architecture and compare it with current Multi-instances scalability architecture. The comparison includes both the analytical queuing model, and the implementation characteristics of each architecture. Analytical analysis shows that Elastic VM implies less response time compared with Multi-instances architecture. Moreover, implementation analysis shows that Elastic VM reduces the provisioning overhead and results in less violation of SLOs. Furthermore, it scales applications, such as Databases, with lower cost and complexity.

Our immediate future work is to compare both architecture empirically against different web application patterns and workload demand. Also, we study optimizing the packing of Elastic VMs into physical hosts. Moreover, we study the Elastic VM architecture's effect on the pricing model in the cloud.

References

[1] "Architecting for the Cloud: Best Practices." [Online]. Available: http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

[2] "Xen hypervisor." [Online]. Available: http://www.xen.org/

[3] "VMWare." [Online]. Available: http://www.vmware.com/

[4] P. Johnson and T. Marker, "Data Center Energy Efficiency Report Product Profile," 2009.

[5] Amazon, "Amazon Elastic Compute Cloud." [Online]. Available: http://aws.amazon.com/ec2/

[6] "GoGrid." [Online]. Available: http://www.gogrid.com/

[7] Y. D. Chess, J. L. Hellerstein, S. Parekh, and J. P. Bigus, "Managing Web server performance with AutoTune agents," *IBM Systems Journal*, vol. 42, no. 1, pp. 136–149, Jan. 2003.

[8] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh, "Online Response Time Optimization of Apache Web Server," pp. 461–478, 2003.

[9] D. Wiese, G. Rabinovitch, M. Reichert, and S. Arenswald, *Autonomic tuning expert*, ser. CASCON '08. New York, New York, USA: ACM Press, 2008.

[10] D. N. Tran, P. C. Huynh, Y. C. Tay, and A. K. H. Tung, "A new approach to dynamic self-tuning of database buffers," *ACM Transactions on Storage*, vol. 4, no. 1, pp. 1–25, May 2008.

[11] W. Iqbal, M. N. Dailey, and D. Carrera, "SLA-Driven Dynamic Resource Management for Multi-tier Web Applications in a Cloud," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington: IEEE, 2010, pp. 832–837.

[12] G. P. Bhuvan Urgaonkar, "An analytical model for multi-tier internet services and its applications," in *In Proc. of the ACM SIGMETRICS2005*, 2005, pp. 291—-302.

[13] A. Dubey, R. Mehrotra, S. Abdelwahed, and A. Tantawi, "Performance modeling of distributed multi-tier enterprise systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, p. 9, Oct. 2009.

[14] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, *Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments*. IEEE, Jun. 2008.