Scaling in Cloud Environments

DOMINIQUE BELLENGER, JENS BERTRAM, ANDY BUDINA, ARNE KOSCHEL, BENJAMIN PFÄNDER, CARSTEN SEROWY
Faculty IV, Department of Computer Science
University of Applied Sciences and Arts Hannover
Ricklinger Stadtweg 120, 30459 Hannover
GERMANY
{dominique.bellenger, jens.bertram1, andy.budina, benjamin.pfaender, carsten.serowyg}@stud.fh-hannover.de, arne.koschel@fh-hannover.de

IRINA ASTROVA
InVision Software OÜ
Lõõtsa 2A, 11415 Tallinn
ESTONIA
irinaastrova@yahoo.com

STELLA GATZIU GRIVAS, MARC SCHAAF
Institute for Information Systems
University of Applied Sciences Northwestern Switzerland
Riggenbachstrasse 16, 4600 Olten
SWITZERLAND
{stella.gatziugrivas, marc.schaaf}@fhnw.ch

Abstract: - This paper describes two approaches to scaling in cloud environments – semi-automatic (also called by request) and automatic (also called on demand) – and explains why the latter is to prefer. Semi-automatic scaling is illustrated with an example of Amazon Elastic Compute Cloud, whereas automatic scaling is illustrated with an example of Amazon Web Services Elastic Beanstalk.

Key-Words: - Scaling, Cloud computing, Amazon Elastic Compute Cloud (Amazon EC2), Amazon Web Services Elastic Beanstalk (AWS Elastic Beanstalk), Experiments

1 Introduction

Cloud computing [1] can be defined as an abstraction of services from infrastructures (i.e. hardware), platforms and applications (i.e. software) by virtualization of resources. To designate these different forms of cloud computing services, three terms have been used: IaaS, PaaS, and SaaS, which stand for Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Service, respectively (see Fig. 1).

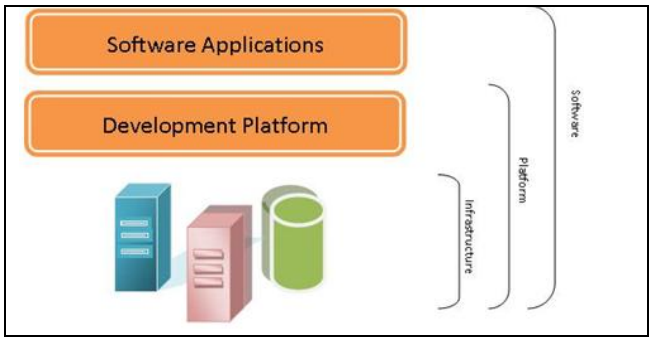

Fig. 1. Cloud computing services [2].

Whereas most IaaS and PaaS providers reveal more or less detailed information on how scaling is done in their products, there are almost no details available from SaaS providers. On the one hand, we can argue that SaaS providers do not disclose how scaling is done in their products because this information is not important for users. What is important for users is that their applications do scale automatically. On the other hand, an automatic scaling mechanism can be viewed as a trade secret and therefore, not revealed by SaaS providers in detail. That is why in this paper we consider scaling on IaaS and PaaS levels only. Scaling on IaaS level is illustrated with an example of Amazon Elastic Compute Cloud (Amazon EC2) [6], whereas scaling on PaaS level is illustrated with an example of Amazon Web Services Elastic Beanstalk (AWS Elastic Beanstalk) [7].

The main contribution of this paper is to present the results of our experiments on an automatic scaling mechanism of AWS Elastic Beanstalk.

ISBN: 978-1-61804-019-0

2 Motivation

Cloud computing shifts the location of resources to the cloud (a metaphor for the Internet) to reduce the costs associated with *over-provisioning* (i.e. having too many resources), *under-utilization* (i.e. not using resources adequately) and *under-provisioning* (i.e. having too little resources). It also reduces the time required to provision resources to minutes, allowing applications to quickly scale both up and down, as the workload changes. Therefore, cloud computing is particularly well suited for applications with a variable workload that experience hourly, daily, weekly or monthly variability in utilization of resources. One example of such applications is online shops, which have to handle their peak loads at Christmas time. Another example is airline booking websites, which have to handle their peak loads during campaigns.

In traditional (i.e. non-cloud) environments, over-provisioning and under-utilization can hardly be avoided [1]. There is an observation that in many companies the average utilization of application servers ranges from 5 to 20 percent, meaning that many resources like CPU and RAM are idle at no-peak times [3].

On the other hand, if the companies shrink their infrastructures to reduce over-provisioning and under-utilization, the risk of under-provisioning will increase. While the costs of over-provisioning and under-utilization can easily be calculated, the costs of under-provisioning are more difficult to calculate because under-provisioning can lead to a loss of users and zero revenues [3].

3 Scaling

Scalability [22] can be defined as the ability of an application to make optimum utilization of resources at different workload levels (i.e. avoiding over-provisioning, under-utilization and under-provisioning).

3.1 Manual Scaling in Traditional Environments

In traditional environments, scalability is achieved by predicting peak loads, then purchasing, setting up and configuring the infrastructure that can handle these peak loads. Since resources are provisioned statically (i.e. at deployment time) and manually (e.g. by adding an application server to the infrastructure), the biggest problem with scaling in traditional environments is high latency. Due to this problem, the mean time until resources are provisioned can be long, which is often the main reason for the unavailability of an application at peak loads [4]. Another big problem is manual monitoring of resources.

Fig. 2 shows manual scaling in traditional environments.

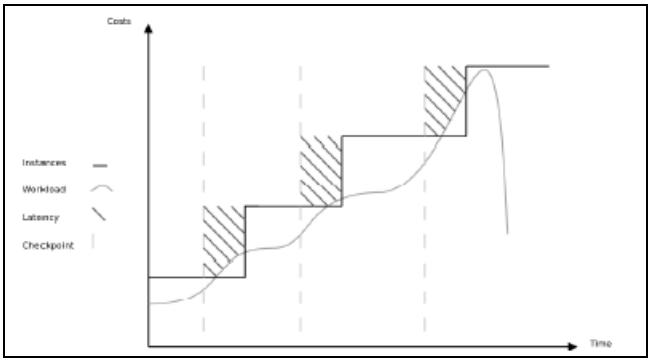

Fig. 2. Manual scaling in traditional environments [4].

3.2 Semi-automatic Scaling in Cloud Environments

In cloud environments, resources are virtualized. This virtualization enables elasticity of the cloud, meaning that the cloud can easily and quickly be resized to adjust to a variable workload. In particular, in cloud environments, resources are provisioned dynamically (i.e. at runtime), automatically (i.e. without user intervention), infinitely and almost immediately (i.e. within minutes and not hours, days, weeks or months like in traditional environments).

Fig. 3 shows semi-automatic scaling in cloud environments. The mean time until resources are provisioned is short, so peak loads can be intercepted in many cases. However, manual monitoring of resources is still necessary. In particular, users are forced to make a tradeoff between requesting more resources to avoid under-provisioning and requesting fewer resources to avoid over-provisioning and under-utilization. Since resources are provisioned *by request*, the problem of the unavailability of an application at peak loads is not completely eliminated.

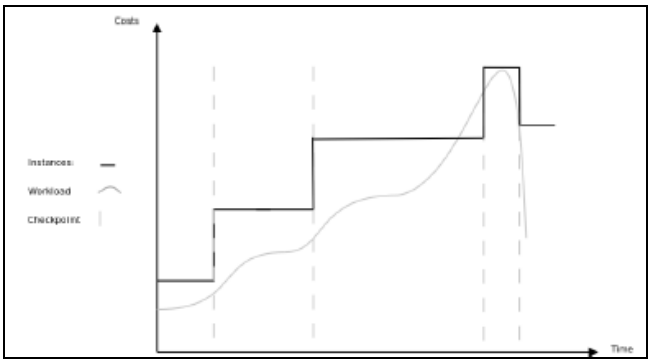

Fig. 3. Semi-automatic scaling in cloud environments [4].

3.3 Automatic Scaling in Cloud Environments

Automatic scaling enables users to closely follow the workload curve of their applications, by provisioning resources *on demand*. With automatic scaling, users can ensure that the number of resources their applications are utilizing automatically increases during demand spikes to handle peak loads and automatically decreases during demand lulls to minimize costs so that users are not forced to pay for the resources they do not need.

But what can be done if the peak loads are so enormous that an application is still unavailable despite automatic scaling? It would be nice if an automatic scaling mechanism could predict these peak loads and provision the needed resources in advance, making the impression of elasticity of the cloud. One technique to achieve this is to identify repeating workload patterns [4]. Another is to use a Markov chain [5].

Fig. 4 shows automatic scaling in cloud environments. It is similar to semi-automatic scaling. The key difference is that manual monitoring of resources is needed no longer.

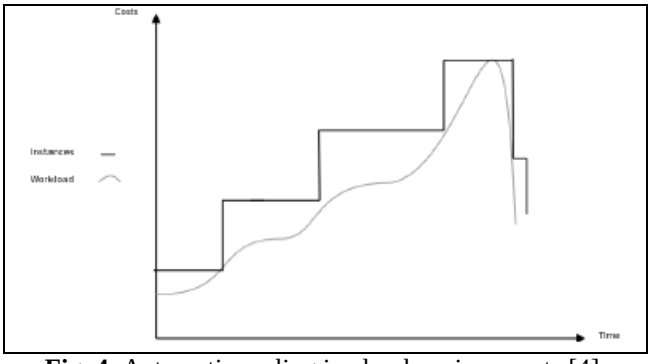

Fig. 4. Automatic scaling in cloud environments [4].

4 Amazon

Amazon is one of the major players in providing cloud computing services like IaaS and PaaS. It offers many products, including Amazon EC2 and AWS Elastic Beanstalk.

4.1 Amazon EC2

Amazon EC2 allows users to rent virtual machines from Amazon and thus, to avoid the costs of purchasing and setting up their own infrastructures.

4.1.1 Instances

The basis of Amazon EC2 is so-called instances. An *instance* is a virtual machine equipped with the specified amount of computing power (including CPU and RAM). Users can monitor for CPU and RAM utilization, and launch or terminate instances when needed. In addition, users can install their applications on instances and bundle these instances to machine images, and then store the machine images in Amazon Simple Storage Service (S3) [8] so later they can easily and quickly launch instances from the machine images, without having to configure the virtual machines again [9].

Fig. 5 shows the life cycle of instance. An instance can be in one of the following states: running, terminated and stopped. Initially, the instance is not running. When a launch command is issued, a new virtual machine is provided. Users can specify how powerful the virtual machine will be with regards to computing power. After the virtual machine is ready for the use, the machine image is deployed to the instance. The instance is now in the running state. From this state, the instance can be brought to the stopped state, by issuing a stop command. The instance in the stopped state can be brought to the running state again, by issuing a start command. This is much more quickly than launching a new instance. But it requires data to be stored on Amazon Elastic Block Store (EBS) [10]. Finally, the instance can be terminated, by issuing a terminate command. The termination will release the resources held by the instance. All the information previously stored in the instance gets lost [11]. If there are data, which should be available after the termination, they should be stored on EBS.

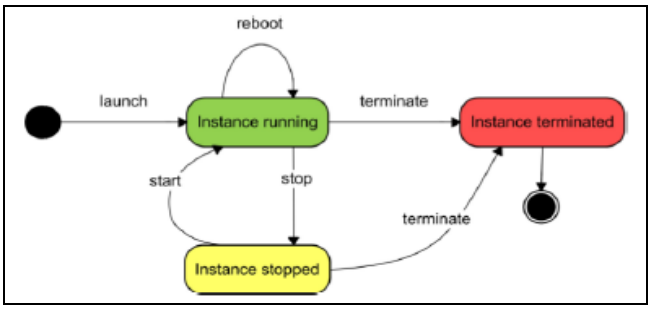

Fig. 5. Life cycle of Amazon EC2 instance [11].

4.1.2 Auto-scaling groups

Users can group instances offering an identical service (i.e. running the same application) into an *auto-scaling group*.

4.2 AWS Elastic Beanstalk

A downside of Amazon EC2 is that it requires users to make many efforts in order to deploy their applications into the cloud. AWS Elastic Beanstalk goes a step further, by providing users with a platform for easy and quick deployment of their applications into the cloud.

For automatic scaling, AWS Elastic Beanstalk uses the following web services: Elastic Load Balancing [12], Auto Scaling [14] and Amazon CloudWatch [13], which works in conjunction with Amazon Web Services Management Console (AWS

ISBN: 978-1-61804-019-0

Management Console) [19], a web-based user interface.

4.2.1 AWS Management Console

AWS Management Console allows users to configure an automatic scaling mechanism of AWS Elastic Beanstalk. For example, users can specify how many instances can and must run (i.e. the maximum and minimum number of running instances) (see Fig. 6). Such configuration will define what to scale, how to scale and when to scale.

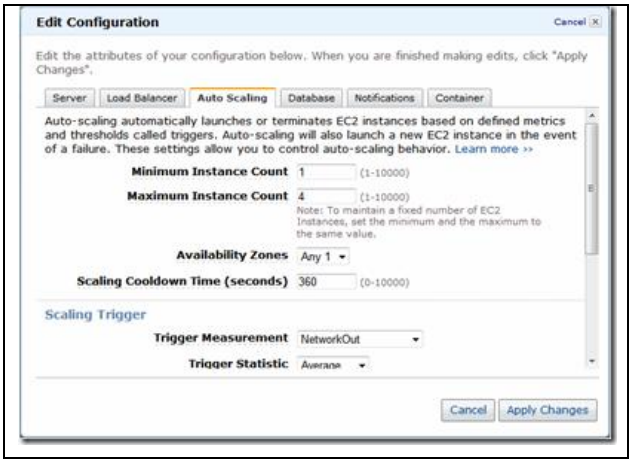

Fig. 6. Configuring AWS Elastic Beanstalk via AWS Management Console [20].

4.2.2 Amazon CloudWatch

Amazon CloudWatch tracks and stores per-instance metrics, including request count and latency, CPU and RAM utilization. Once stored, the metrics can be visualized using AWS Management Console (see Fig. 7). With AWS Management Console, users can get real-time visibility into the utilization of each of the instances in an auto-scaling group, and can easily and quickly detect over-provisioning, under-utilization or under-provisioning.

However, Amazon CloudWatch was primarily developed for the use with Elastic Load Balancing and Auto Scaling.

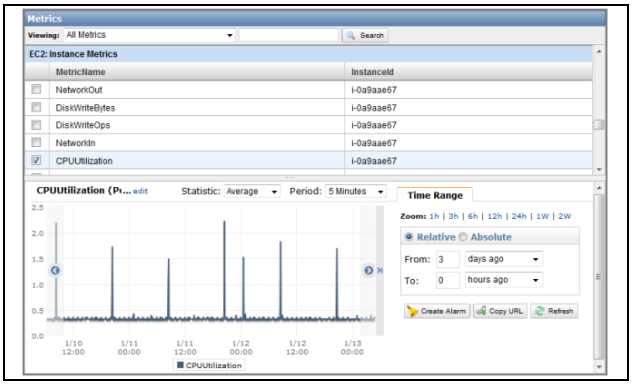

Fig. 7. Visualizing Amazon CloudWatch metrics via AWS Management Console [21].

4.2.3 Elastic Load Balancing

Elastic Load Balancing enables a load balancer, which automatically spreads load across all running instances in an auto-scaling group based on metrics like request count and latency tracked by Amazon CloudWatch. If an instance is terminated, the load balancer will not route requests to this instance anymore. Rather, it will distribute the requests across the remaining instances.

Elastic Load Balancing also monitors the availability of an application, by checking its "health" periodically (e.g. every five minutes). If this check fails, AWS Elastic Beanstalk will execute further tests to detect the cause of the failure. In particular, it checks if the load balancer and the auto-scaling group are existing. In addition, it checks if at least one instance is running in the auto-scaling group. Depending on the test results, AWS Elastic Beanstalk changes the health status of the application.

The possible values for the health status are:

- *Green*: The application has responded within the last minute.
- *Yellow*: The application has not responded within the last five minutes.
- *Red*: The application has not responded for more than five minutes or another problem was detected by AWS Elastic Beanstalk (e.g. the load balancer is not available anymore).
- *Gray*: The status of the application is unknown (e.g. the application is not ready for the use yet).

4.2.4 Auto Scaling

Auto Scaling automatically launches and terminates instances based on metrics like CPU and RAM utilization tracked by Amazon CloudWatch and thresholds called *triggers*. Whenever a metric crosses a threshold, a trigger is fired to initiate automatic scaling. For example, a new instance will be launched and registered at the load balancer if the average CPU utilization of all running instances exceeds an upper threshold.

Auto Scaling also provides fault tolerance. If an instance reaches an unhealthy status or terminates unexpectedly, Auto Scaling will compensate this and launch a new instance instead, thus assuring that the specified minimum number of instances are running constantly.

5 Experiments

Using Amazon Eclipse plug-in [15], we created a sample project called MyTravelLog. This project was a ready-to-deploy application (i.e. WAR archive); it consisted of servlets, property files, Java libraries like J2EE and the application specific libraries like

ISBN: 978-1-61804-019-0

Spring. We used the project to experiment with the automatic scaling mechanism of AWS Elastic Beanstalk. The goal of our experiments was to know how quickly the application can automatically scale both up and down in response to load.

In our experiments, we largely assumed the default values for the Elastic Load Balancing settings. This means that we deployed the application to an Amazon micro-EC2 instance with one virtual CPU and approximately 600 MB RAM.

To measure the mean time until an instance launches or terminates when a trigger is fired, we set all monitoring time intervals of Amazon CloudWatch to the minimum possible values (viz. one minute). Furthermore, we configured Auto Scaling as follows: if the CPU utilization value is above the average of 80 percent, one instance should launch. Next if the CPU utilization value is below the average of 60 percent, one instance should terminate. Also, we specified that there should be at least one and at most four running instances at any point in time. (Elastic Load Balancing was configured automatically.)

Also, we developed a servlet, which just ran through a while-loop for several seconds. This servlet was called in many parallel requests to create a variable workload.

Fig. 8 shows the automatic scaling of the created workload as well as the mean time until instances are launched or terminated. At first, there was only one running instance, which contained the deployed application; the CPU utilization was between 5 and 9 percent. After five minutes, we started five parallel requests every five seconds, which caused the CPU utilization to rise from 5 percent to almost 100 percent on the only running instance. In about three minutes, the automatic scaling mechanism detected that the CPU utilization was above 80 percent, fired a trigger and launched a new instance. After several additional minutes, the health monitoring of Elastic Load Balancing detected an unhealthy status of the application, changed the health status from green through yellow to red and launched yet a new instance. The first additional instance was ready in about six minutes after the trigger had been fired. The CPU utilization went down to about 65 percent. The launching process of the second instance completed in three minutes later and the CPU utilization went down to about 45 percent. After five additional minutes, the automatic scaling mechanism detected that the CPU utilization was below 60 percent and there were three instances running; so one of those instances was terminated.

Thus, our experiments showed that the application can automatically scale up when the load increases and automatically scale down when the load decreases. The mean time until an instance is ready for the use is between three and six minutes (which is a very good result).

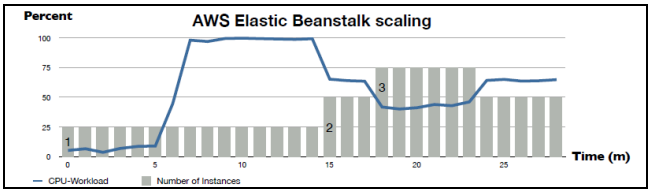

Fig. 8. Results of experiments on AWS Elastic Beanstalk.

6 Conclusion

We described the automatic scaling mechanism of AWS Elastic Beanstalk and experimented with it. Our experiments showed that provisioning resources took a few minutes.

However, the automatic scaling mechanism of AWS Elastic Beanstalk is somehow hidden behind the platform. As a downside, users might have to adapt their applications to the platform characteristics and the requirements for scalability.

Amazon provides a best-practices guide [16] on how users should develop their applications for the best fit for cloud environments. The most important guidelines are: an application should be divided into loosely coupled components that can be distributed across different application servers and executed in parallel. Furthermore, the application should be as stateless as possible [17]. If a component fails or is temporarily not available, the application should continue to run. This can be achieved by developing the component as self-rebooting and using a message queue. If the component is temporarily not available, messages will be stored in the message queue and delivered later when the component is available again.

Other platforms like Google App Engine [18] also let users have their applications to automatically scale both up and down according to demand but with even more restrictions on how users should develop their applications. That is why we selected AWS Elastic Beanstalk for our experiments.

A downside of AWS Elastic Beanstalk is that currently it does not provide any web service to predict demand for the near future. Theoretically, the statistical usage data of the last few months or years could be used to predict time intervals during which more or fewer resources are needed.

References:

[1] C. Braun, M. Kunze, J. Nimis, and S. Tai. Cloud Computing, *Web-based Dynamic IT-Services*. Springer Verlag, Berlin, Heidelberg, 2010.

[2] Demystifying SaaS, PaaS and IaaS, http://e2enetworks.com/2010/05/03/demystifying-saas-paas-and-iaas/

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010

[4] J. Yang, J. Qiu, and Y. Li. A profile-based approach to just-in-time scalability for cloud applications. *Proceedings of the IEEE International Conference on Cloud Computing* (CLOUD 2009), Washington, DC, USA, pages 9–16, 2009. IEEE Computer Society

[5] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. *CNSM*, pages 9–16, 2010

[6] Amazon.com. Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/

[7] Amazon.com. AWS Elastic Beanstalk, http://aws.amazon.com/elasticbeanstalk/

[8] Amazon.com. Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/s3/

[9] T. Chieu, A. Mohindra, A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment, *Proceedings of the IEEE International Conference on E-Business Engineering*, pages 281–286, 2009

[10] Amazon.com. Amazon Elastic Block Store (Amazon EBS), http://aws.amazon.com/ebs/

[11] U. Koch, and O. Hunte. Konzepte & Beispiele zu Cloud Computing. Bachelorarbeit, Fachhochschule Hannovee, Fakultat IV, Abteilung Informatik, 2010

[12] Amazon.com. Elastic Load Balancing, http://aws.amazon.com/elasticloadbalancing/

[13] Amazon.com. Amazon CloudWatch, http://aws.amazon.com/cloudwatch/

[14] Amazon.com. Auto Scaling, http://aws.amazon.com/autoscaling/

[15] Amazon.com. AWS Toolkit for Eclipse, http://aws.amazon.com/eclipse/

[16] J. Varia. Architecting for the cloud: best practices. http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

[17] P. Marshall, K. Keahey, and T. Freeman. Elastic site: Using clouds to elastically extend site resources, *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, pages 43–52, 2010

[18] Google. Google App Engine, http://code.google.com/appengine/

[19] Amazon.com. AWS Management Console, http://aws.amazon.com/console/

[20] T. Anderson. Amazon's Elastic Beanstalk auto-scales your cloud application, http://www.itwriting.com/blog/3691-amazons-elastic-beanstalk-auto-scales-your-cloud-application.html

[21] Amazon.com. Get a Single Metric for a Specific EC2 Instance, http://docs.amazonwebservices.com/AmazonCloudWatch/2010-08-01/DeveloperGuide/index.html?CHAP_TerminologyandKeyConcepts.html

[22] J. Cáceres, L. Vaquero, L. Rodero-Merino, Á. Polo, and J Hierro. Service Scalability over the Cloud, *Handbook of Cloud Computing*, eds. B. Furht and A. Escalante, Springer Verlag, Berlin, Heidelberg, 2010