Infrastructure-as-a-Service (IaaS) with Amazon EC2/Eucalyptus

Florian Fittkau

Christian-Albrechts-University Kiel
Department of Computer Science
24098 Kiel, Germany

Abstract. Cloud computing, for instance provided by Eucalyptus, becomes more and more popular because it facilitates the construction of scalable applications. However, the horizontal scaling is not solely exhausted by web application in a cloud. Additional techniques for horizontal scaling are required. This paper describes Eucalyptus and Amazon EC2, and investigates the performance of the online-adaptation framework SLAstic utilizing Eucalyptus for automatic scaling of an example web application. The paper presents a performance evaluation of SLAstic utilizing Eucalyptus incorporating a fixed number of nodes scenario and a varying number of nodes scenario. It shows that web applications running in Eucalyptus can automatically scale up and down with the usage of SLAstic. The results are demonstrated on the basis of JPetstore with a simulated typical day-night-cycle load for web applications.

1 Introduction

Nowadays, many software service providers have to buy server hardware to serve the demand of their customers. This hardware has to be laid out for the peak rates of the workload. At times where the peak rate is not reached, the server hardware is idle, which is not economic. Cloud computing can help to solve this problem. When the user demand for the service increases, the cloud can provide more server instances and when it drops, some server instances can be terminated for saving money. Thus, the cloud provides the possibility for applications to dynamically scale up and down.

In the cloud, this horizontal scaling becomes more important than the performance of a single application instance. In a cloud with Infrastructe-as-a-Service, the horizontal scaling is achieved by starting and stopping virtual machine instances. For this purpose, Amazon [1] provides a feature named Auto Scaling. Auto Scaling reacts to abnormal usage of resources and runs new server instances if needed. If server instances are idle, Auto Scaling terminates those instances. Like most open-source cloud software, Eucalyptus [8] lacks the feature of automatic scaling. This paper shows how SLAstic [29] can be used to emulate Auto Scaling for web applications in Eucalyptus.

The remainder of the paper is structured as follows. Section 2 outlines cloud computing and its different characteristics, service models, and deployment

models. Then, Section 3 describes a selection of web services provided by Amazon with focus on Elastic Compute Cloud (EC2). The following Section 4 shows an open-source alternative to Amazon EC2, named Eucalyptus. The subsequent Section 5 presents a performance evaluation of SLAstic utilizing Eucalyptus. For the performance evaluation an example web application, namely JPetStore [10], is used. Afterwards, Section 6 describes the related work. Finally, Section 7 concludes the results of the paper and presents future work.

2 Cloud Computing

Cloud computing is a relatively new term. Hence, many definitions for cloud computing exist. Vaquero et al. [27] have surveyed different cloud definitions. Here, we use the National Institute of Standards and Technology (NIST) definition by Mell and Grance [12] because this definition becomes a de-facto standard. Subsection 2.1 describes cloud computing in general. Then, the NIST definition that is divided into the three parts essential characteristics, service models, and deployment models, is described in Subsections 2.2 to 2.4.

2.1 Overview

Cloud computing brings different advantages as described by Armbrust et al. [3].The first advantage is the illusion of infinite computing resources available when ever needed. Thus, cloud computing users do not need to plan if there will be peaks in the client demands which may be hard to forecast or even be impossible if the user demand is unknown. The second advantage is the elimination of an up-front commitment. Hence, cloud computing enables clients to start a small business without a huge investment in hardware. The third advantage is the ability to pay for the use of computing resources on a short-term bases as needed, resulting in an elastic way of using the hardware.

Many commercial providers like Amazon provide a cloud service. Many open-source solutions for cloud computing have been developed, too. Endo et al. [7] present them in their survey. One of these is Eucalyptus which is described in Section 4.

2.2 Essential Characteristics

The NIST definition for cloud computing defines five essential characteristics that a service must fulfill in order to be a cloud service. These are listed and described below.

1. On-demand self-service A user can rent computing capabilities like storage and computing time. This can be done automatically without human interaction of the service provider.

2. Broad network access The capabilities can be accessed over the network by standard mechanisms that are available on heterogeneous platforms like mobile phones and laptops.

3. Resource pooling The provider's computing resources are pooled to serve multiple clients. The clients have no exact knowledge where the physical or virtual resources are allocated.

4. Rapid elasticity Resources can be rapidly and elastically allocated to enable quick scale up and down. The client can purchase the virtually unlimited resources in any quantity at any time.

5. Measured Service The cloud system automatically controls and optimizes the used resources by monitoring the usage. For the provider and clients, transparency is provided by monitoring, controlling and reporting the resource usage data.

2.3 Service Models

The cloud providers can offer their service at different levels of abstraction with regard to configuration and programming options. These are described by others as the architecture of the cloud [4, 11, 17, 18]. The different kinds of service models are described below.

Infrastructure-as-a-Service (IaaS) Infrastructure-as-a-Service provides the lowest level of abstraction with a maximum of configuration options compared to the other service models. In this service model the client setups and runs instances of virtual machine images. Hence, the client can create the full software stack by himself. The virtual machine images can be provided by the cloud provider or can be created by the client himself, for instance. Most IaaS provider offer additional services for common client demands like storage. A popular cloud provider with IaaS is, for instance, Amazon with EC2, which is described in Section 3.

Platform-as-a-Service (PaaS) In PaaS, the provider defines and maintains the programming environment for the client. In most cases only specific programming languages with even more constraints to meet the environment specifications are supported. The client can access the computational resources through language-specific APIs and libraries. Popular examples for this type of service model are Google App Engine [9] and Microsoft Azure [13].

Software-as-a-Service (SaaS) SaaS brings the highest level of abstraction with no configuration options apart from the rented software. In this service model the client rents access to the software in the cloud. The software can then be used over the Internet. The client advantages from no installation, no maintenance of the program, and guarantee of the latest available software patch. Google Docs or Microsoft Office Live are examples for SaaS.

2.4 Deployment Models

Clouds can be deployed with four different deployment models. These are public clouds, private clouds, hybrid clouds, and community clouds.

Public Clouds The cloud infrastructure can be accessed by the general public, such that everyone can rent and use the computing resources. For example, Amazon provides a public cloud described further in Section 3.

Private Clouds For a company public clouds can have disadvantages. First of all, the sensitive business data must be transfered into and kept in the public cloud resulting in possible security issues. Secondly, the public cloud provider can go bankrupt or the service can be down for a long period. To avoid those situations a company can build its own private cloud operated by itself or a third party. An open-source cloud software for this purpose is, for example, Eucalyptus described in more detail in Section 4.

Hybrid Clouds Considering this type of deployment model both private and public providers are used by a client. This is often used by companies to exploit the advantages of public and private clouds. The privacy-critical applications are kept in a private cloud and the resource-intensive and uncritical applications are run in a public cloud.

Community Clouds The last deployment model is a community cloud. This kind of cloud can be built by many different providers to give a special community access to the computing resources.

3 Amazon Web Services

Amazon's Elastic Compute Cloud (EC2) is a popular public IaaS cloud. Subsection 3.1 describes EC2 and Subsection 3.2 outlines a selection of other web services provided by Amazon.

3.1 Amazon Elastic Compute Cloud (EC2)

The service enables running EC2 instances. Those instances can be started by several ways. The client can directly use the web service API, or he can utilize console tools developed by Amazon, or he can start EC2 instances through a web GUI, named AWS Management Console. For running an EC2 instance the user chooses an Amazon Machine Image (AMI) to start. Amazon provides different pre-assembled images. However, the user can create his own images by configuring a pre-assembled image, for example. The started instances are non persistent by default. Thus, the user has to care for saving his data by using EBS, for instance.

Due to the trade secret the architecture of EC2 is not published by Amazon. Thus, only a few things are known about the architecture of EC2 [28]. Xen is the hypervisor for the images. Amazon's payment model is a pay per use model. Everyone with a credit card can rent storage and virtual machines with different types that are made of different configurations of CPU cores and main memory. These types are named small to extra large. The more resources a type offers the more it costs per hour. The small type has currently 1,7 GB RAM, 1 EC2 Compute Unit (1 virtual Core), 160 GB local instance space and is a 32-Bit platform. 1 EC2 Compute Unit maps to 1.0 - 1.2 GHz of a 2007 Opteron or 2007 Xeon Prozessor. A small instance with Linux spawned on a server in Ireland costs currently 0.095 USD per hour. Amazon provides the possibility to specify the region of the real hardware that is running the virtual machines. This can be an advantage when a client has to fulfill a regulatory framework. One region is, for example, Europe. A region has at least one availability zone. Availability zones are isolated against failures that occur at other availability zones.

3.2 Further Services

Amazon provides different additional web services [2]. Although Amazon provides more services, we only focus on the services for persistence and the service Auto Scaling, which SLAstic should emulate, here.

Auto Scaling Amazon provides Auto Scaling as a feature for automatic horizontal scaling. In combination with CloudWatch, which monitors the resource utilization on the different EC2 instances, Auto Scaling provides dynamic starting of new EC2 instances and termination of unused EC2 instances. The user can configure Auto Scaling for his needs, e.g. a maximum number of running instances.

Elastic Block Store (EBS) The Elastic Block Store can be used for persistent data storage for EC2 instances. A raw, unformatted block device can be assigned to any EC2 instance to save data beyond shutting down of the instance.

Relational Database Service (RDS) RDS provides a relational database that is automatically installed and started on an extra EC2 instance, such that the user does not need to install, maintain, and backup the database. A small database instance currently costs 0.11 USD per hour.

SimpleDB SimpleDB is a simplified database management system, which lacks different kind of features like joins. The service is suited for highly available persistence with large amount of data.

Simple Storage Service (S3) Simple storage service stores files on a key-value basis. Amazon introduced a bucket concept for this service. Every bucket must be globally uniquely named, so that the user can access the bucket over a domain provided by Amazon. A bucket can take files with a size up to 5 TB each. It is possible to choose the location of the servers that should store the files. US, Europe, and Asia are possible locations. The files can be downloaded with HTTP, SOAP, REST, and BitTorrent.

4 Eucalyptus

Eucalyptus was designed at the University of California, Santa Barbara, by Nurmi et al. [14, 16] and is an open-source IaaS cloud software. Eucalyptus stands for "Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems" [20]. Eucalyptus was written for research purposes because it is not easily possible to experiment with a specific cloud infrastructure with services like Amazon's EC2 not under own control. Subsection 4.1 describes the architecture of Eucalyptus.

4.1 Architecture

Eucalyptus focuses on the IaaS level for two reasons [15]. First, the developers wanted it to be interface-compatible to Amazon's EC2 for testing and exchangeability purposes. Second, the developers claim that other layers like PaaS and SaaS can be layered upon IaaS, which can be beneficial for other research.

Eucalyptus utilizes Xen and KVM virtualization technologies [6]. The developers provide Linux tools, named euca2ools, for accessing the cloud from command line. However, it is possible to directly call the web service API.

Eucalyptus was designed to meet the environment found at most universities. For the most part, these are compute clusters that are not directly connected to each other. Figure 1 illustrates this fact and shows the three-level hierarchical architecture of Eucalyptus. The Cloud Controller and Walrus provide access from a global point. The Cloud Controller has connection through a public network to the different Cluster Controllers. The Cluster Controllers then have access to the machines in their private network. That way, only the Cluster Controllers need access to the public network.

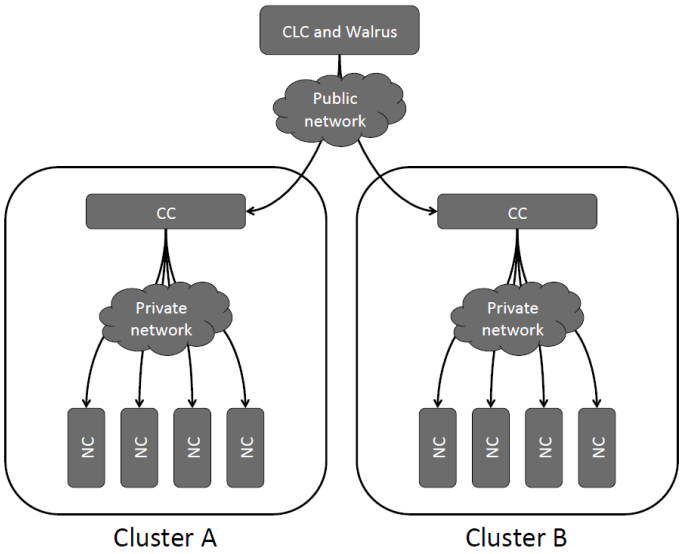


Fig. 1. Eucalyptus architecture taken from Nurmi et al. [16]

Cloud Controller (CLC) Every Eucalyptus deployment has only one Cloud Controller (CLC). The CLC polls the Cluster Controllers for meta data about the running instances and resources. In addition, the CLC receives the requests from the user and distributes the commands issued by the user to the Cluster Controllers based on the received meta data.

Storage Controller (Walrus) When an instance is terminated, all modifications and data of the terminated instance are lost. For persistent storage, Walrus can be used. Walrus is the emulation of Amazon's S3.

Cluster Controller (CC) This is the controller for a subnetwork of computation nodes. By this concept only the Cluster Controller needs public access and the other machines in their private network can be shielded against access from outside the cluster. However, in this concept everything must be passed through the CC.

Node Controller (NC) The Node Controller runs on every physical computation node to start, configure, and terminate virtual machine image instances. Furthermore, the node sends descriptive data to the CLC through the CC.

5 Performance Evaluation of SLAstic Utilizing Eucalyptus

In this section the performance of SLAstic for automatic scaling of web applications in an Eucalyptus deployment is evaluated. Subsection 5.1 describes SLAstic and the made Eucalyptus adaptation. Then, Subsection 5.2 points out the goals of the performance evaluation. Afterwards, Subsection 5.3 illustrates the experimental setting. Subsection 5.4 presents the two scenarios for the evaluation, namely one experiment with a fixed number of nodes without SLAstic and one experiment with a varying number of nodes incorporating SLAstic. Then, Subsection 5.5 presents the results of the evaluation and Subsection 5.6 discusses the results.

5.1 SLAstic and Eucalyptus Adaptation

The online-adaptation framework SLAstic provides architectural runtime reconfiguration. It employs changing component deployments and server allocations at runtime. With the dynamic changing of component deployments, the performance and resource efficiency of component-based software systems can be controlled and changed in an elastic manner, while the software is running.

We extended SLAstic to function with Eucalyptus by implementing an interface provided by SLAstic. SLAstic calls the interface methods when a reconfiguration should take place. This can be, for instance, the allocation of a new node. In those methods, we implemented calls to the euca2ools. Additionally, we adapted a pre-assembled Ubuntu image with the software for a web server with JPetStore and uploaded the image into our Eucalyptus deployment. SLAstic is configured with the ID of the image such that new instances are started from this image.

5.2 Goals

The experiment should verify that SLAstic can be used to enable automatic horizontal scaling of web applications in Eucalyptus. In spite of automatic scaling, the average response times of the web application should be below an acceptable threshold of 100 ms in 95 percent of the time, which is typical for most web applications. Furthermore, the average CPU utilization of allocated nodes should be higher with a varying number of nodes than with a fixed number of nodes.

5.3 Experimental Setting

Figure 2 shows the deployment of the used components. Our test environment incorporates three nodes, namely the Eucalyptus cloud, the SLAstic node, and the workload generator node. In the Eucalyptus cloud, instances of the worker image can be spawned. The worker image is the adapted image mentioned in

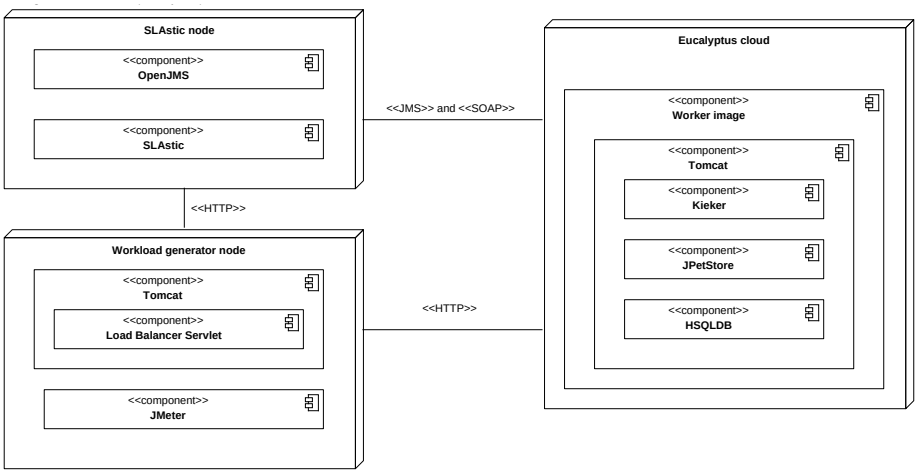


Fig. 2. Deployment of the used components

Subsection 5.1 and includes a Tomcat servlet container 6.0.18 [22] with JPetStore 5.0, its own database (HSQLDB 1.8 [23]), and Kieker 1.3 snapshot [26]. Our JPetStore is adapted with an additional computation in every request such that each request takes around one ms computation time. Kieker is included for monitoring the CPU usage and executions of the annotated methods in JPetStore. It is configured to send the monitored data every 15 seconds to the OpenJMS 0.7.7 beta [24] queue on the SLAstic node. SLAstic 0.01a is deployed on the SLAstic node, analyzing the data from the OpenJMS queue and calculating if a new instance of the worker image has to be allocated or an instance can be released. The probabilistic and intensity-varying workload is generated by JMeter 2.4 [21] with Markov4JMeter [25] on the workload generator node. The JMeter profile is configured to fetch the destination IPs from the load balancer servlet, running on the same node, on the start of a new use session. The servlet manages a list with running instances of the worker image and passes a random IP to JMeter. The server list is updated by SLAstic. We do not use the Tomcat load balancer because we want a more flexible way of adding and removing instances of the worker image through web service calls.

Table 1 presents the hardware configuration for the used m1.small image type. SLAstic starts instances of the worker image with this type. The two AMD Opteron 2384 processors provide 8 CPU cores in sum. Thus, the maximum number of instances is limited to 8. To show the horizontal scaling effect by having the maximum number of instances possible the CPU cores per instance are set to 1. Each instance gets 1024 MB of DDR2-667 RAM allocated. The server has a 1 Gigabit network connection.

Maximum number of instances	8
CPU cores per instance	1
RAM per instance	1024 MB
CPU type	2x AMD Opteron 2384 with 2.7 GHz
RAM type	DDR2-667
Network	1 Gigabit

Table 1. Our Eucalyptus configuration

5.4 Scenarios

The experiment will include two scenarios. The first scenario with a fixed number of nodes without SLAstic is the control scenario. In the second scenario, SLAstic will be used for automatic scaling of JPetStore. After describing the used workload intensity, the two former mentioned scenarios are described in detail.

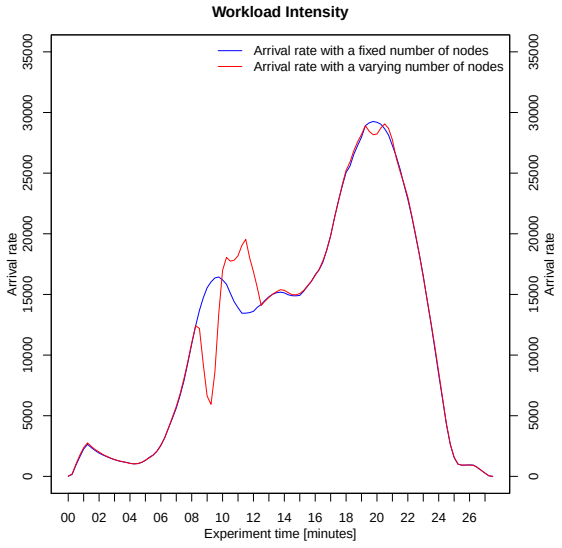


Fig. 3. The used day-night-cycle workload intensity where minutes map to hours of a day

Figure 3 presents the generated workload for both scenarios as derived from the measurement data. The workload intensity function origins from the Thursday-workload of a CeWe-Color-system [19]. It conforms to a typical day-night-cycle workload intensity for most websites. The minutes map to hours of a day. In the

morning the workload intensity increases until there is a first peak at noon and a second higher peak in the evening. Then, the workload intensity decreases until there are only few requests at night.

In spite of using the same workload intensity function in both scenarios, the two curves differ from minute eight to twelve. In the varying number of nodes scenario, JMeter seems to be waiting for the answers from the node and thus the curve shifts by one minute in the described interval.

Scenario I: Fixed Number of Nodes In this scenario six nodes are started at the beginning and are manually registered in the load balancer servlet. Furthermore, the SLAstic node is dismissed from the deployment shown in Figure 2.

Scenario II: Varying Number of Nodes Initially, one instance is running. SLAstic is configured to automatically spawn or terminate nodes based on characteristic lines with a variance interval for smoothing. The first characteristic lines, where the configuration changes from one node to two nodes, is set at an arrival rate of 5,000 requests per minute. After this line, every 5,000 requests per minute a new characteristic line is set. Every characteristic line posses a variance interval of 1,000 requests per minute for smoothing.

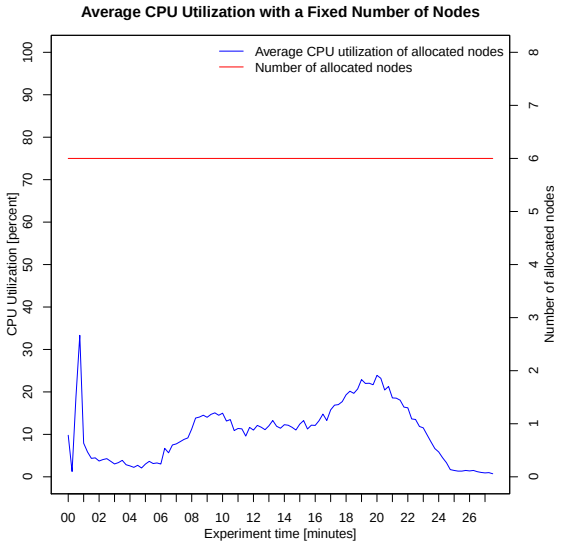
5.5 Results

Figure 4 and 5 show the results of the experiment. This section describes the results for the two scenarios.

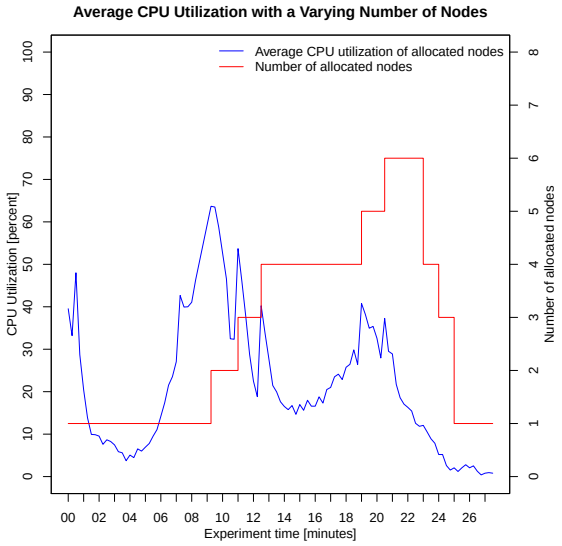
Scenario I: Fixed Number of Nodes In Figure 4(a) the average CPU utilization of allocated nodes in the scenario without SLAstic are presented. In minute one, there is a peak of 30 percent average CPU utilization of allocated nodes. Then, the average CPU utilization drops to seven percent and after that it increases until minute nine, where the second peak of 15 percent is reached. Until minute 20, the average CPU utilization increases to 24 percent. Thereafter, it drops to nearly 0 percent until minute 27.

Figure 5 illustrates the average response times at the entry-level methods for scenario I and II. Until minute one, the average response times are above 15 ms. From minute two to 27 the average response times range between two and five ms, but are constantly below five ms.

Scenario II: Varying Number of Nodes In Figure 4(b) the results of the scenario utilizing SLAstic are shown. There is a 50 percent peak in the average CPU utilization of the one allocated node at the beginning. Then, the average CPU utilization drops to seven percent until minute four. Until minute nine, the utilization increases. At minute nine, there is a peak with 64 percent of average CPU utilization. At this point, the second node is allocated. In minute eleven and 13, the average CPU utilization has lower peaks and the third respectively fourth



(a) Scenario I: Fixed number of nodes



(b) Scenario II: Varying number of nodes

Fig. 4. Average CPU utilization of allocated node

node are allocated. Then, the average CPU utilization increases until minute 19, where the fifth node is allocated. In minute 20 is another peak and the sixth node is started. Afterwards, the average CPU utilization decreases until minute

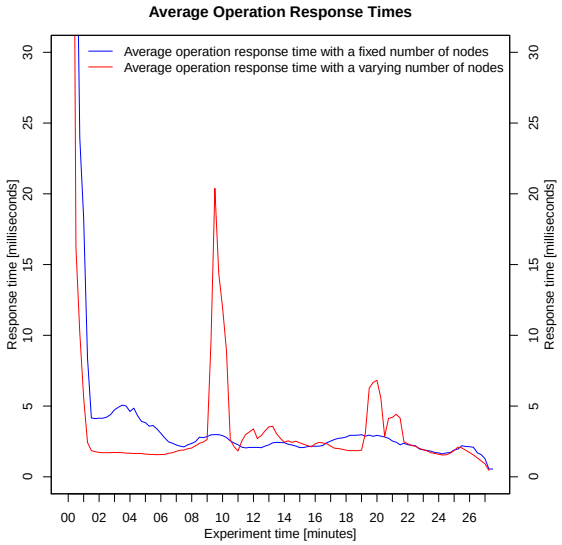


Fig. 5. Average response times

27. With the decrease, the allocated nodes, except the first node, are terminated in minute 23, 24, and 25.

Figure 5 presents the average response times at the entry-level methods. At the beginning the average response times are above 15 ms. Until minute ten, the average response times are below three ms. In minute ten, they increase to 20 ms. Then, the average response times drop and are constantly below four ms until minute 20. In minute 20, there is a small rise to seven ms. Then, the average response times drop again and stay around three ms until minute 27.

5.6 Discussion of the Results

In this section the described results for the two scenarios are discussed and compared to each other.

Scenario I: Fixed Number of Nodes The first peak in the average CPU utilization is caused by the first calls to JPetStore. These calls consume some initialization time. The other increases and decreases in the average CPU utilization accompany with the increases and decreases in the workload intensity.

For the average response times, there is a high peak at the beginning. This is again caused by the first calls to JPetStore, which take some initialization time. After minute two, the average response times are constantly below five ms, which is an acceptable value for average response times.

Scenario II: Varying Number of Nodes The first peak in the average CPU utilization is caused by the first calls to JPetStore like in scenario I. The peaks in minutes nine, eleven, twelve, 19, and 20 result from a new allocation of a node at each point in time. At those points the first calls to the new running JPetStore take again initialization time. The intervals from minute two to nine and minute 14 to 19 show increases, which accompany with the increases in the workload intensity. The interval from minute 23 to 27 shows a decrease in the average CPU utilization, which is caused by the falling workload intensity.

For the average response times, there is, like in the fixed number of nodes scenario, a high peak at the beginning caused by the first calls to JPetStore. The second peak in minute ten with 20 ms average response time results from the allocation of the second node. The first calls to the second node result in higher average response times. The relative high increase is caused by the fact that only two nodes are running and thus the effect is higher in the average metric. The same happens at every small peak. However, the rest of the time the average response times are below five ms. In a real system, where a new node would be started once an hour, the impact of the first calls would be even lower. Thus, the scenario fulfills the requirement for acceptable average response times.

Comparison of the 2 Scenarios In scenario II, the average CPU utilization of allocated nodes increased compared to scenario I. This can be seen, for instance, at minute eight, where the average CPU utilization is 45 percent, while in the fixed number of nodes scenario the average CPU utilization at minute eight is eleven percent. The response times in scenario II are below five ms most of the time, which is the same value as in the fixed number of nodes scenario.

6 Related Work

Wee and Liu [30] have tested the efficiency of different load balancing techniques, which is essential for scaling of websites in the cloud. They measured that hardware-based load balancers scale better than software-based load balancers. However, a hardware-based load balancer might not be available. Other techniques like DNS load balancing and Layer 2 Optimization have shortcomings and are not possible with Amazon EC2, for instance. Thus, Wee and Liu propose client-side load balancing for clouds. They wrote a JavaScript-based script which automatically chooses a back-end web server based on the load from a received server list. This technique scales better than the other techniques except for hardware-based load balancers.

Baun and Kunze [5] evaluated the performance of Amazon EC2 versus Eucalyptus. They compared the storage performance, CPU performance, network transfer rate, and network latency. They conclude that Eucalyptus can provide the same functionality and possibly better performance compared to Amazon Web Services dependent on the hardware on which Eucalyptus is installed.

7 Conclusions and Future Work

The paper described cloud computing using a *de-facto* standard definition by the NIST. Then, Amazon EC2 and Eucalyptus were illustrated. The latter was utilized for a performance evaluation of SLAstic.

The results of the evaluation show that SLAstic can emulate Auto Scaling for web applications in Eucalyptus. The average CPU utilization of allocated nodes is increased, while the average response times are the same as in a fixed number of nodes case most of the time. Thus, SLAstic can be used for automatic horizontal scaling of web applications in Eucalyptus.

However, in most cases a central database is desired. This can lead to a central point, which can be the bottleneck. One strategy for solving this bottleneck can be synchronizing the different databases. Another strategy can be allocating an instance with an extra large virtual machine image type, so that the database can benefit from multiple CPU cores.

References

[1] Amazon.com, Inc. Amazon EC2. `http://aws.amazon.com/de/ec2/`, last visited 2010-10-29.

[2] Amazon.com, Inc. Amazon AWS. `http://aws.amazon.com/de/products/`, last visited 2011-01-14.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[4] J. Barr, T. von Eicken, and E. Troan. Application Architecture for Cloud Computing. `http://www.rpath.com/corp/images/stories/white_papers/WP_ArchitectureForCloudComputing.pdf`, last visited 2011-01-23.

[5] C. Baun and M. Kunze. Building a private cloud with Eucalyptus. In *Proceedings of the 5th 2009 IEEE International Conference on E-Science Workshops*, pages 33–38. IEEE Computer Society, 2009.

[6] E. Caron, F. Desprez, D. Loureiro, and A. Muresan. Cloud computing resource management through a grid middleware: A case study with diet and eucalyptus. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, pages 151–154. IEEE Computer Society, 2009.

[7] P. T. Endo, G. E. Goncalves, J. Kelner, and D. Saok. A Survey on Open-source Cloud Computing Solutions. In *VIII Workshop em Clouds, Grids e Aplicacoes (WCGA 10)*. Sociedade Brasileira de Computacao, 2010.

[8] Eucalyptus Systems, Inc. Eucalyptus. `http://www.eucalyptus.com/`, last visited 2010-10-29.

[9] Google. Google App Engine. `http://code.google.com/intl/de-DE/appengine/`, last visited 2011-01-09.

[10] iBATIS team. JPetstore. `http://mirror.synyx.de/apache/ibatis/binaries/ibatis.java/JPetStore-5.0.zip`, last visited 2010-11-05.

[11] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the Cloud? An architectural map of the Cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009.

[12] P. Mell and T. Grance. The NIST Definition of Cloud Computing. `http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf`.

[13] Microsoft Corporation. Windows Azure Platform. `http://www.microsoft.com/windowsazure/`, last visited 2011-01-09.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus : A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. Technical Report UCSB 2008-10, Computer Science Department, University of California, Santa Barbara, 2008.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: an open-source cloud-computing infrastructure. In *Jornal of Physics: Conference Series 180*, pages 124–131. IOPscience, 2009.

[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *CCGRID '09: Proceedings of the 9th 2009 IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE Computer Society, 2009.

[17] B. P. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proceedings of the 2009 IEEE International Conference on Networked Computing and Advanced Information Management*, pages 44–51. IEEE Computer Society, 2009.

[18] J. W. Rittinghouse and J. F. Ransome. *Cloud Computing: Implementation, Management, and Security.* CRC Press, 2009.

[19] M. Rohr, A. van Hoorn, W. Hasselbring, M. Lübcke, and S. Alekseev. Workload-intensity-sensitive timing behavior analysis for distributed multi-user software systems. In *1st Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW 2010)*, pages 87–92. ACM, Jan. 2010.

[20] T. Tan and C. Kiddle. An Assessment of Eucalyptus Version 1.4. Technical Report 2009-928-07, Grid Research Centre, University of Calgary, Canada, 2009.

[21] The Apache Software Foundation. JMeter. `http://jakarta.apache.org/jmeter/`, last visited 2011-01-04.

[22] The Apache Software Foundation. Tomcat. `http://tomcat.apache.org/`, last visited 2011-01-04.

[23] The hsql Development Group. HSQLDB. `http://www.hsqldb.org/`, last visited 2011-02-07.

[24] The OpenJMS Group. OpenJMS. `http://openjms.sourceforge.net/`, last visited 2011-01-04.

[25] A. van Hoorn, M. Rohr, and W. Hasselbring. Generating probabilistic and intensity-varying workload for Web-based software systems. In S. Kounev,

I. Gorton, and K. Sachs, editors, *Performance Evaluation — Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08)*, volume 5119 of *Lecture Notes in Computer Science*, pages 124–143. Springer, June 2008.

[26] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Technical Report 0921, 2009.

[27] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. In *SIGCOMM Comput. Commun. Rev.*, volume 39, pages 50–55. ACM, 2009.

[28] J. Varia. Cloud Architectures. `http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf`, last visited 2011-01-23.

[29] R. von Massow, A. van Hoorn, and W. Hasselbring. Performance Simulation of Runtime Reconfigurable Component-Base Software Architectures. Technical report, Software Engineering Group, University of Kiel, Germany, 2010.

[30] S. Wee and H. Liu. Client-side load balancer using cloud. In *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 399–405. ACM, 2010.