Investigating the Impact of Deployment Configuration and User Demand on a Social Network Application in the Amazon EC2 Cloud*

Matheus Cunha, Nabor Mendonça and Américo Sampaio
Mestrado em Informática Aplicada (MIA) – Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321, Edson Queiroz, CEP 60811-905 Fortaleza, CE, Brazil
Email: {mathcunha, nabor, americo.sampaio}@unifor.br

***Abstract*—One of the main challenges faced by current users of infrastructure-as-a-service (IaaS) clouds are the difficulties to estimate cloud resources according to their application needs. Even though cloud platforms are elastic and provide fast means to acquire or release resources it is important to understand the best ways to do that considering a diversity of providers with many different resource types and prices. This work reports on an experimental investigation conducted on a popular cloud benchmark based on a social network application running on top of the Amazon EC2 cloud. Our experiments aim at finding cost-effective ways to select the different EC2 resource types and deployment configurations based on the demand imposed to the application (measured in number of simultaneous users) and identify the configuration that gives the best return in terms of its cost per user.**

I. Introduction

Cloud computing offers an attractive business model that is based on the provision of on-demand, pay-per-use resources (e.g., computing, storage, applications and platforms), usually in the form of Internet services [1], [2]. Several cloud computing solutions already exist, ranging from open source platforms that enable the creation of private clouds inside an organization [3], [4], to commercial cloud providers, such as Microsoft's Windows Azure [5], Amazon's EC2 [6], Google's AppEngine [7], Salesforce [8] and Rackspace [9]. EC2 and Rackspace, in particular, are typical examples of the so called infrastructure-as-a-service (*IaaS*) clouds [1], offering virtual servers (i.e., virtual machines with a specific capacity for CPU, memory and storage) and virtual storage that can be created, accessed and managed remotely as services via public application programming interfaces (APIs). In this way, customers of these commercial cloud providers have an easy way of acquiring their required computing resources without having the burden to manage the physical infrastructure.

However, for most cloud customers the wide variety of resources offered by an increasing number of commercial IaaS cloud providers brings a new challenge, which is to be able to select the best cloud resources according to the needs and characteristics of their applications. For example, in the case of the EC2 cloud, the customer has a set of more than 10 virtual machines (or instances) types with varied levels of computational capacity and prices [6]. Therefore, developers need to have a deep understanding of the requirements of their applications as well as of the expected performance of the different cloud resources and services that each provider offers in order to make a wise selection.

Even though cloud computing resources are elastic and can be acquired on demand, the task of estimating the optimal amount of resources to match the application's expected workload is still a very difficult challenge. This problem, if not properly addressed, can result in either an *overprovisioning* situation, where more resources are acquired than what is really necessary, or in *underprovisioning*, where the acquired resources are not enough to handle the application workload [1]. Both situations normally result in financial loss either because one is paying for resources that are not being used or because users may give up on the application when it no longer handles the load.

In this regard, some fundamental questions that are commonly asked in these scenarios are: (i) What types of cloud resources and/or deployment configurations are the most effective (from the perspective of both performance *and* cost) to a given demand (measured, for instance, in the number of concurrent users accessing the application)?; (ii) In case the demand drops below a certain threshold, which are the best configuration options?; (iii)What is the most cost-effective configuration choice: to concentrate the load on one high-capacity instance or to distribute it among multiple instance of small or medium capacity?

Our work aims at investigating answers for these questions, particularly under lightweight workload conditions, as they are very relevant for current cloud customers. To do this we selected a well-known cloud benchmark [10] (described in section II), so that we could investigate the relation between the demand imposed on a social network application (a very common cloud application) running on the Amazon EC2 cloud (the most popular IaaS cloud provider). Our experiments (described in section III) show how the cost per user varies with the demand imposed to the application and how the choice for EC2 instance types and configurations reflects on the costs paid by the cloud customer. In the paper we also compare our work with relevant related work (section IV) and outline our future research agenda (section V).

* This work is partially funded by Fundação Edson Queiroz, Universidade de Fortaleza, Projeto OW2.

II. Cloud Benchmark

The Cloudstone project [10] offers a very useful benchmark for conducting experiments in IaaS clouds that has been successfully used for tests in Amazon EC2. Cloudstone is composed of a Web 2.0 social-event calendar web application (Olio) that runs on two popular platforms (Ruby on Rails and PhP). Moreover, a workload generator (Faban) is also provided that allows to run different tests on the Olio application and collect data from different executions (e.g., varying the number of concurrent users). All tools are open source and enable the user to easily change the application architecture (e.g., creating multiple application servers). This flexibility of architectural change and the open source nature of the tools were key drivers for our choice of Cloudstone as the selected benchmark for our research.

A. Olio Application

The Olio application[1] provides two basic set of features: (i) *User Management*, which allow users to create an account in order to be able to use the main features of the application; and (ii) *Event Management*, which allow registered users to create events, attend events or add comments to an event. Users are also able to browse events (by date or tag) as well as to see maps of event locations.

The Olio architecture on the Ruby on Rails platform, which is the platform we chose for our experiments, is organized in three layers: web server, application server, and database. The web server layer contains the static content of the web application running either in Apache or Nginx[2] and serves as load balancer. The application server layer contains the business logic of the application running on the multithreaded Thin Ruby application server[3]. Moreover, this layer can be configured as a cluster and have multiple application server instances. Finally, the database layer is responsible for the persistence of the application data in a MySQL database that can also be configured with a memory-based cache mechanism for faster access.

B. Faban Load Generator

Faban[4] is an open-source Markov-chain, closed-loop, session-based workload generator that can be used for testing Web 2.0 applications [10]. The framework defines operations, transactions and associated statistics collection and reporting that enables one to easily test interactive web applications such as Olio. For example, it gathers data (e.g., response time) from various simultaneous users issuing different Olio operations (e.g., addUser, addEvent, postComment).

Faban allows to configure some parameters for each test run such as the number of concurrent users. During each test Faban collects the response time for each request made by the load generator (from the time it is issued until the last byte of response is received). A successful test can be considered as, for example, the one where at least 90 percent of the responses arrive within the maximum response time threshold. This information was used in our experiments as a measurement of how well the test application performed in the cloud, as described in the next section.

[1]http://incubator.apache.org/projects/olio.html

[2]http://wiki.nginx.org/Main

[3]http://code.macournoyer.com/thin/

[4]http://faban.org

EC2 instances	**Price (US$/hour)**	**Relative cost (w.r.t. smaller instances)**
t1.micro	0.02	—
m1.small	0.085	4.25 x t1.micro
c1.medium	0.17	2 x m1.small 8.5 x t1.micro
m1.large	0.34	2 x c1.medium 4 x m1.small 17 x t1.micro
m2.xlarge	0.50	1.47 x m1.large 2.94 x c1.medium 5.88 x m1.small 25 x t1.micro
c1.xlarge m1.xlarge	0.68	1.36 x m2.xlarge 2 x m1.large 4 x c1.medium 8 x m1.small 34 x t1.micro
m2.4xlarge	2.00	2.94 x c1(m1).xlarge 4 x m2.xlarge 5.88 x m1.large 11.76 x c1.medium 23.53 x m1.small 100 x t1.micro

TABLE I
A sample of the Amazon EC2 instance types and prices.

III. Experiments

We ran all our experiments on the Amazon EC2 cloud. That decision was due to the fact that EC2 is both a technical and commercial reference in the current IaaS cloud landscape. Since Amazon charges EC2 resources according to the *geographical region* in which they are created, with each geographical region offering slightly different prices, we chose to run all our experiments in the region with the lowest prices, i.e., the *EC2 US East* region.

EC2 provides a number of distinct types of virtual machines (instances), which differ in terms of memory and processing capacity, operating system, pricing model, amongst other characteristics [6]. Table I lists several of the instance types available at the EC2 US East region, along with their price (in US$ per hour) and relative cost for the Linux/UNIX operating system option. Note that the highest priced instance type shown in Table I (*m2.4xlarge*) is 100 times more expensive than the lowest priced one (*t1.micro*)! Such diversity of resource configurations and costs means that choosing a particular EC2 instance type (or a collection of instance types) to run a particular application requires not only a detailed knowledge of the difference instance types and price models available, but also a deep understanding of how those different instance types may (or may not) match the application's expected demand.

For our experiments we decided to focus on relatively lightweight workloads. Our motivation stems from our belief that most commercial web applications follow typical workload distributions with regular periods of very low demand (usually during non-business hours such as in the evenings, weekends and public holidays); therefore, it is important to understand how to best reduce or release the application' resources so as to incur a minimum cost to the cloud customer during these periods.

Given that most exiting work on cloud performance evaluation consider application demands in the order of thousands of simultaneous users [10], in our work we opted to evaluate Olio's performance with the number of concurrent users varying from 25 (very low demand) to 600 (moderate demand). Each experiment consisted of hosting Olio in the Amazon cloud using different EC2 instance types and configurations, and then testing the application performance under that demand range using the Faban load generator. Each unique test involving a different deployment configuration and a different number of users was executed three times. This was meant to identify potential variations in the results, which may happen due to the well-known performance instability of EC2, as previously reported elsewhere [12]. Each test lasted about 800 seconds, and was considered successful if the application performance matched the 90 percent acceptance threshold defined by Faban.

It is important to note that all results reported in this paper only reflect the moment in which they were generated, since the performance of a particular EC2 instance may be influenced by several factors, including the types and number of other instances sharing the same physical server [6].

A. Experiment 1: Investigating single server configurations

In this experiment we used three EC2 instances. The first one was used to run the Faban load generator; the second one the MySQL database; and the third one the Olio application server, Thin, and the NGINX load balancer. Our goal with this experiment was to investigate how Olio would handle our defined workload when deployed in single application server configurations using different EC2 instance types. We also wanted to understand the impact of each chosen configuration in terms of the application's overall cost. To this end, we fixed the types of the load generator and the MySQL database instances to the same type used in the Cloudstone project, namely *c1.xlarge* [10], and varied the type of the application server instance according to the 8 EC2 instance types listed in Table I.

The results of this experiment are shown in the first eight rows of Table II. As we can see, with 25 concurrent users (the lowest demand level investigated) the Olio application server can be hosted in a single instance of type *t1.micro*, the lowest priced EC2 type. However, with this instance type the application matched its demand in only one out of the three tests executed, which shows the instability of the EC2 cloud for low-end instance types. In this regard, hosting the application server in an instance of type *m1.small* would provide more stable results, for a slightly higher cost. Moving to the next demand levels, from 50 to 150 concurrent users, we can see that all instance types satisfied the demand, with the exception of the type *m1.large*, which did not fully match the demand with 150 concurrent users.

Now moving to 200 concurrent users, we can see from Table II that that the lowest priced instance type that can match that specific demand level is *m2.xlarge*. However, this particular type only partially matched that demand. Therefore, to improve application performance stability without a significant increase in cost, an instance of type *c1.xlarge* would be recommended. We call attention to the fact that different instance types with similar prices may offer different performance results, as it was the case with types *m1.xlarge* e *c1.xlarge* within this particular demand range. The reason is that some instance types are specifically tailored to a given application profile (for instance, more memory or processing intensive). In the particular case of this experiment, we can see that Olio has a more processing intensive profile, since it benefits the most from instance types that favor processing over memory capacity. It is also evident that paying more for higher-end instance types does not necessarily translate into better cost/performance results. This is clearly visible in the results for the *m2.4xlarge* and *c1.xlarge* instance types, with the former costing about three times the price of the latter but offering a slightly more unstable performance when tested under demand levels of 300 concurrent users and above. Overall, if we consider the wide price variation amongst the different EC2 instance types, its a foregone conclusion that an adequate choice of resource configuration at these demand levels might result in substantial monetary gains to the cloud customer.

Once we have a better understanding of the relation between the EC2 resource configurations and the demand levels they support, it is possible to further analyze how effectively those configuration can be used from an economical perspective. In this way, it would be easier for one to understand the impact of choosing a particular instance type configuration to host the target application in terms of its overall cost. To do this we calculate the *cost per user* associated with Olio for the different EC2 configurations that have successfully matched the different demand levels investigated. The cost per user metric was originally proposed as part of the Cloudstone project to evaluate the cost-effectiveness of cloud applications, and is defined as the rate between the cost to host the application in the cloud, using a particular resource type and/or configuration, and the maximum number of users effectively supported by that configuration [10].

To give an example of how this metric was calculated in our experiments, consider the cost per user associated with Olio for the *m1.small* instance type. Since the maximum number of concurrent users supported by that type in our tests was 25, its cost per user was calculated by dividing its price (US$ 0,085) by 25, yielding a hourly cost of US$ 0,0034 per user. For the other instance type configurations that supported different demand levels, we calculated their cost per user for each supported demand level separately.

Figure 1 depicts the cost per used calculated for Olio,

App. server instances	Demand (in number of simultaneous users)										
	25	50	75	100	125	150	200	300	400	500	600
Single server configurations											
t1.micro	⊙	○	○	○	○	○	○	○	○	○	○
m1.small	●	○	○	○	○	○	○	○	○	○	○
c1.medium	●	●	●	●	●	●	○	○	○	○	○
m1.large	●	●	●	●	●	⊙	○	○	○	○	○
m2.xlarge	●	●	●	●	●	●	◎	○	○	○	○
c1.xlarge	●	●	●	●	●	●	●	●	◎	○	○
m1.xlarge	●	●	●	●	●	●	●	○	○	○	○
m2.4xlarge	●	●	●	●	●	●	●	●	⊙	○	○
Multiple servers configurations											
c1.medium (x2)	●	●	●	●	●	●	●	●	○	○	○
c1.medium (x3)	●	●	●	●	●	●	●	●	●	●	●

Legend	
●	Application fully matched user demand (3/3 successful executions)
◎	Application partially matched user demand (2/3 successful executions)
⊙	Application sporadically matched user demand (1/3 successful executions)
○	Application did not match user demand (0/3 successful executions)

TABLE II
OLIO'S PERFORMANCE RESULTS USING SINGLE AND MULTIPLE APPLICATION SERVER CONFIGURATIONS.

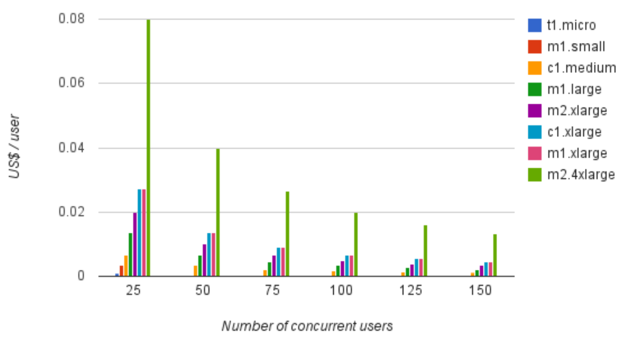

Fig. 1. Olio's cost per user using single application server configurations, under low demand levels.

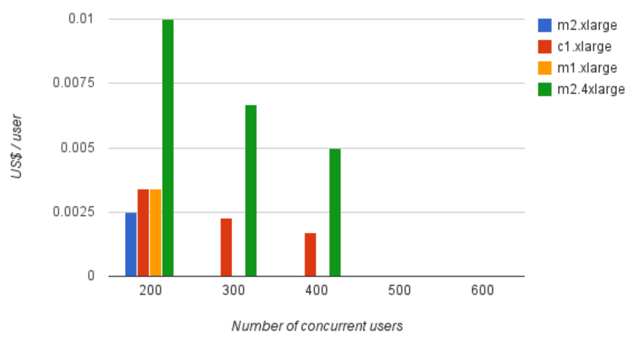

Fig. 2. Olio's cost per user using single application server configurations, under moderate demand levels.

configured with a single application server using different EC2 instance types, under demand levels of 25 to 150 concurrent users. As we can see, for 25 concurrent users there are significant differences in the cost per user associated with each of the eight instance types evaluated, varying from US$ 0,0008 (type *t1.micro*) to US$ 0,08 (type *m2.4xlarge*). This means that, under that particular demand, choosing the highest-end instance type over the lowest-end one would cost 100 times more with comparable performance results. Also, this cost difference between the lowest and the highest priced instance types that support that demand starts to decrease as the number of concurrent users increases. With 150 users, the highest priced instance type (*m2.4xlarge*) is "only" 11,76 times more expensive than the lowest priced one (*c1.medium*). Note that, for most of the levels within this demand range (from 50 to 150 concurrent users), the instance type that offers the best cost/benefit is *c1.medium*.

Let us now consider the more moderate demand levels, from 200 to 600 concurrent users, whose cost per user results are depicted in Figure 2. With exact 200 concurrent users, the best configuration choice is *m2.xlarge*. Note that, since both *c1.xlarge* and *m1.xlarge* have the same price, their cost per user at that demand level is also the same. However, as the demand level reaches 300 concurrent users, the two lowest priced memory intensive instance types (i.e., *m2.xlarge* and *m1.xlarge*) no longer appear in graph, as they both fail to support that level of demand. Instance type *m2.4xlarge*, in turn, successfully supports up to 400 concurrent users, but is never the recommended choice given its high price and the fact that there is at least one lower priced instance type that also supports the same demand levels. Overall, the most cost-effective instance type for such moderate levels of demand is *c1.xlarge*, which once again confirms the processing intensive profile of the Olio application.

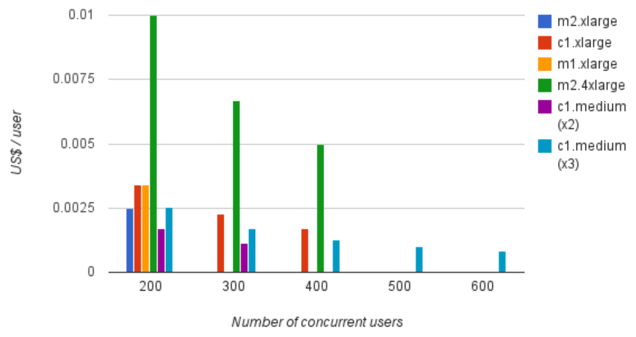

Fig. 3. Olio's cost per user using single and multiple application server configurations, under moderate demand levels.

B. *Experiment 2: Investigating multiple servers configurations*

Our goal with this second experiment was to investigate whether deploying Olio using EC2 configurations with multiple application server instances would improve its performance and/or its cost per user, when compared to the other single server configurations described in the previous subsection, particular under more moderate demand levels. To this end, we once more fixed the instance types of the load generator and the MySQL database to *c1.xlarge*. In addition, we investigated two new configurations in which the Thin application server was deployed in clusters of two and three instances of type *c1.medium*, respectively. We decided to deploy cluster with this particular instance type due to its relative lower price (compared to the other high-end instances) and the fact that it has offered the best cost-benefit among all instance types evaluated up to 150 concurrent users.

Olio's performance results using multiple application server configurations are shown in the last two rows of Table II. As we can see, the two new configurations, with two and three *c1.medium* instances, respectively, effectively support up to 300 concurrent users. However, the configuration with only two *c1.medium* instances fails to support any demand beyond that level, while the configuration with three instances of this type can support up to 600 concurrent users. This contrasts favorably with the numbers reported in our first experiment, in which the most effective single application server configurations (the ones with instance types *c1.xlarge* and *m2.4xlarge*) only supported up to 400 concurrent users, and even so with unstable results.

The cost per user results calculated for the two new multiple application server configurations, along with those of the most robust single application server configurations, under the more moderate demand levels (from 200 to 600 concurrent users), are depicted in Figure 3. We can see in that figure that up to 300 concurrent users the most economical choice is offered by the configuration with two *c1.medium* instances. This choice changes to the configuration with three *c1.medium* instances when the number of concurrent users increases from 400 to 600. Compared to the configuration that uses a single *c1.xlarge* instance, which was identified as the best single application server configuration choice for these levels of demand, the new configuration with three *c1.medium* instances is not only cheaper (three *c1.medium* instances cost, together, about 25% less than a single *c1.xlarge* instance) but also more efficient (supporting 50% more concurrent users).

Demand (# users)	**Recommended EC2 configurations**
25	m1.small
50	c1.medium
75	c1.medium
100	c1.medium
150	c1.medium
200	c1.medium (x2)
300	c1.medium (x2)
400	c1.medium (x3)
500	c1.medium (x3)
600	c1.medium (x3)

TABLE III
OLIO'S RECOMMENDED APPLICATION SERVER CONFIGURATIONS.

Finally, Table III lists the set of recommended EC2 instance types and configurations to host the Olio application server, considering the low (25 concurrent users) to moderate (600 concurrent users) demand levels investigated in our work. By looking at that set of configurations and the other results discussed throughout this section, we can draw some very interesting conclusions regarding the deployment of Olio in the Amazon EC2 cloud:

- Application server deployments using only high-capacity instances are not cost-effective to handle periods of low and moderate demand;
- Clustered application server configurations based on medium-capacity instances are a viable option at moderate demand levels (from 200 to 600 concurrent users); and
- Scaling down the application even during periods of very low demand can result in substantial monetary savings for the cloud customers.

IV. Related Work

There are several works focused on assessing different aspects of cloud computing. Some works focus on measuring the performance of IaaS clouds. Cloudsleuth [11] offers a tool for measuring performance of a reference application running on the Amazon EC2 cloud (US East region). The reference application is a web-based product catalog that allows to advertise and view products. The tool issues requests that simulate users from all over the world (more than 168 countries) as well as collects and compares the different response times. The results show that the response times can be very varied specially because of the latency of the client side issuing the requests. For some countries (e.g., Africa and Asia) the response time is much higher than others (e.g., North America) due to bad internet service and distance to the cloud.

Similarly, Bitcurrent[5] also analyzes the performance of commercial clouds from the perspective of the end user. However its analysis is more complete than Cloudsleuth as it analyzes several cloud providers (different EC2 regions, GoGrid, Google App Engine and Joyent) as well as measures not only performance but also availability. The performance results corroborate the ones from Cloudsleuth and show that geographic regions that are distant from the cloud or that provide poor internet services generate higher response times than others. Regarding availability its interesting finding is that while most clouds provide a high degree of availability (all IaaS providers are above 93 percent) the ISPs that represent the client side can vary from as low as 10 percent to as high as 98 percent on average by country.

Both works described above are very important but have a different focus to our work because they do not analyze costs. Regarding the study of costs in IaaS clouds, one interesting work is the CloudCmp [12] approach. CloudCmp offers a tool that enables to compare performance and cost of a common set of services (computing, storage and networking) offered by different public cloud providers. They devised a common set of general metrics in order to analyze and compare each service provider and their results point out that IaaS providers had significant variations of performance during the tests. Their cost metrics focus on giving a general idea of the different service prices according to the capacity offered (e.g., number of cores for computing service). Therefore, this is different to our work which focused on studying the costs from the perspective of application workloads.

Finally the work described in [10] shows an interesting study based on the Cloudstone benchmark. The study shows results of running different configurations of the Olio application in Amazon EC2. They use different architectural configurations of the Olio application (varying the platform between Ruby on Rails and PhP) as well as the database layer (with or without caching) and collect data of the maximum number of users that succeed on the benchmark tests. The key differences from our study are: (i) only two EC2 instance types (*m1.xlarge* and *c1.xlarge*) were selected while we study eight instance types including more limited instances such as *t1.micro* and *m1.small*; (ii) the costs for different loads are not studied while our study can provide the cost per user for each instance type based on load; (iii) our study also considers more options for configuring the business layer as a cluster of two or three application servers and compares the costs with several different single instance deployments.

V. Conclusion

As we have advocated in this paper, choosing adequate resource configurations to effectively host a particular cloud application can have a direct impact on the application's infrastructure cost. Therefore, using cloud resources without having a detailed knowledge of their potential cost/benefits may bring the risk of misprovisioning, ultimately resulting in financial losses to the application owners. Therefore, before a cloud application is deployed, it is crucial to understand which resource configurations would more effectively match the application's computational needs and how those configurations could be adapt to meet future changes in the application's demand, two definitively non-trivial tasks which may be the source of great concern to most cloud consumers.

In this paper we have described the method and results of a preliminary investigation in which we have analyzed the impact of using different resource configurations, under a variety of workload conditions, on the cost-effectiveness of a social network application deployed in the Amazon EC2 cloud. Our results show that there are significant cost and performance differences among the spectrum of EC2 resource configurations investigated, and that cluster-based configurations using intermediate resources may outdo single-server configurations using high-end resources in terms of both performance *and* cost. We expect our results to be useful as a first step towards a more systematic approach to help cloud users in planning their cloud resources according to their applications' requirements and needs. In this regard, we envision several lines of future work, including: improving the automation level of our analysis tools; performing more test with cloud applications of different domains, possibly deployed on different cloud providers; and extending our findings to support auto-scaling of cloud applications, both horizontally (by adding or removing resources to/from a given configuration) and vertically (by replacing a configuration's resources with other resources of higher or lower computational capacity).

[5]http://www.bitcurrent.com

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08.* Ieee, 2009, pp. 1–10.

[3] Eucalyptus, "Eucalyptus the open source cloud plataform," 2009, http://open.eucalyptus.com/.

[4] OpenNebula, "Open source tookit for cloud computing," 2010, http://www.opennebula.org.

[5] Azure, "Windows azure platform," 2010, http://www.microsoft.com/windowsazure.

[6] EC2, "Amazon elastic compute cloud," 2010, http://aws.amazon.com/ec2.

[7] AppEngine, "Google app engine," 2010, http://code.google.com/appengine/.

[8] S. Force, "Sales force," 2010, http://www.salesforce.com/.

[9] Rackspace, "The rackspace cloud," 2009, http://www.rackspacecloud.com/.

[10] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. of CCA.* Citeseer, 2008.

[11] CloudSleuth, "Cloudsleuth," 2011, https://www.cloudsleuth.net/web/guest/home/.

[12] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th annual conference on Internet measurement.* ACM, 2010, pp. 1–14.