PERFORMANCE ANALYSIS OF A JAVA WEB APPLICATION RUNNING ON AMAZON EC2

Gábor IMRE, Hassan CHARAF, László LENGYEL
Department of Automation and Applied Informatics, Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics
Magyar tudósok krt. 2., Budapest H-1117, Hungary, email: gabor, hassan, lengyel@aut.bme.hu

ABSTRACT

More and more companies decide to deploy their new or existing applications at cloud computing providers. A few of the most important forces driving this process are the scalability, business agility and mobility offered by the cloud environments. A number of issues, however, tend to inhibit the adoption of cloud based systems, e.g. security, compliance, vendor lock-in, or inconsistent performance. This paper focuses on the performance questions of a web application implemented on Java platform, deployed at the servers of Amazon Web Services, which is considered one of the dominant cloud providers currently. We present the results of a load testing that investigates the throughput and response time of the application on two different types of instances (Micro and Small) with different computing power and price. We quantify the effect on the user-perceived performance when choosing either of these instance types.

Keywords: *cloud computing, performance analysis, load testing, web applications*

1. INTRODUCTION

Cloud computing offers companies a completely different paradigm for operating their applications and IT infrastructure as opposed to traditional server rental. Upfront hardware and software costs can be avoided thanks to the "Pay-As-You-Go" principle of the cloud providers. The computing capacity can be easily scaled to extremely high levels according to the growing needs. In the case of traditional server rental, overprovisioning of the capacity is needed so that peek loads can be handled without drastic drops of performance that lead to dissatisfied users. In a cloud environment, however, the costs of overprovisioning can be avoided, since scaling down is also possible for periods with lower client load.

Thanks to these advantages, several papers report more and more extensive adoption of cloud computing. The survey of NorthBridge [1] states that 75% of the responders think that they will develop software using a Platform-as-a-Service [2] , which is a specific type of cloud computing, until 2017. According to the forecast of Gartner [3] we can expect an annual growth rate of 17.7% in the end-user spending on public cloud services from 2011 through 2016. Also, a survey among chief financial officers [4] shows that 54% of the responders think that half of their transactions will be supported by Software-as-a-Service [2] systems running on cloud infrastructure no later than 2015. Despite the enthusiasm, some worries and drawbacks of cloud computing are known as well. In [5], the authors list ten of these obstacles, including availability, data lock-in, data confidentiality and auditability, and unpredictability of performance.

Our paper focuses to the latter one, more specifically we investigate the performance of two different server types in the Amazon Elastic Compute Cloud (Amazon EC2) [6]. Amazon EC2 offers a wide variety of server types (or instance types), each with different CPU and IO capacity, memory size, etc. Naturally, higher capacity comes with higher price. The cheapest instance type is the so called Micro instance, out of which one instance can be used even for free for one year. Amazon recommends this instance type for websites with lower traffic.

Our previous work [7] investigated the performance of a Micro instance in EC2, we tried to explore the load levels that a Micro instance can serve with an acceptable performance. We examined different database sizes and different levels of client load. We found, that in most of the considered cases, the average response times are less than 2 seconds with the Micro instance, which can be deemed acceptable. With larger database size and client load, however, the response time can grow to ineligible large. Also, the response time can be extremely high even at smaller spikes in the traffic.

In this paper, we extend our scope on EC2 Small instances, whose computing capacity and cost differ from that of Micro instances, as we describe in Section 2. Small instances are still recommended for websites with lower traffic. The main questions, we are looking for the answers to in this paper are: how does a Small instance perform compared to a Micro instance? What is the effect on the response time of the application if we choose a Small instance instead of a Micro instance? How should we decide among them? Answering these questions is crucial if we want to optimize the operating costs of our systems. For this reason, we executed some load tests against an open source web application, the Apache Roller [8] blog engine. The application was once deployed on an Amazon EC2 Micro instance, and once on a Small instance so that we can compare the results.

The rest of the paper is organized as follows. The necessary background information about the Amazon EC2 platform, the characteristics of the investigated server instances and the application under test is described in Section 2. Section 3 contains the main contributions: the measurement methodology, the hardware and software environment of the measurements; the results of the measurements are presented and discussed here, as well. Section 4 summarizes related work. Finally, we draw conclusions in Section 5.

2. BACKGROUNDS

This section describes the application under test, the Amazon EC2, and the Micro and Small server instance types that we investigate.

The web application under test was Apache Roller, an open source weblog engine written in Java. It can be deployed on the most popular Java application servers (e.g. Tomcat, GlassFish, JBoss) and can be used with several vendors' databases (e.g. MySQL, Derby, PostgreSQL). It offers a large number of features, like group blogging, moderation, spam prevention, RSS 2.0 and Atom 1.0 support and a built-in search engine with own indexing built on Apache Lucene [9]. The layout and style of the blogs, the caching and rendering system are highly customizable. From our point of view, it is important that it is widely used at sites with several thousands of users and blogs, like the very popular developer blog of DZone [10]. This lets us assume that the application itself does not have any implementation issues causing a performance bottleneck. We note that the application has several configurable properties that affect the performance, like caching, indexing of the comments, etc. However, we did not tweak these settings, we kept the default values for all of them.

We decided to deploy the Apache Roller application in the Amazon EC2 platform, because it is the earliest and most dominant provider among the Infrastructure-as-a-Service (IaaS [11]) solutions. It is important to note that Amazon offers Platform-as-a-Service (PaaS [11]) services as well, like different types of storage solutions, message queues or e-mail sending, however, we did not use any of them, since we wanted to investigate an existing application without any PaaS-specific modifications.

With Amazon EC2, we can use virtual server instances with a large variety of operating systems, on top of which we can install any kind of applications. Amazon EC2 offers several types of server instances, each with different CPU capacity, memory size and disk I/O performance [12]. The instances can be purchased in three options.

- The *On-demand* option means that the usage of the server is billed by the hour, this way an instance costs nothing as long as it is not started.

- In case of the *Reserved* option, you have to pay the price of the instance one or three years upfront, loosing some flexibility, but achieving a lower hourly price.

- The *Spot instances* represent a very dynamic pricing option, meaning that the customers can bid on the unused capacity of Amazon EC2. The Spot instances can run as long as the bid exceeds the current Spot price that changes periodically based on supply and demand.

Amazon offers a one-year-long free trial period as well, during which one can run On-demand Amazon EC2 Micro instances 750 hours each month. Since they are On-demand instances, the customer can decide if he runs one instance continuously in each month ($31 * 24 = 744$), or runs multiple instances for shorter periods of time. Besides the constraint on the instance type, the free trial period has other limitations as well, like $30GB$ of storage, 2 million I/Os and 15 GB bandwidth out per month. The hardware specifications of Micro instances [13] state that a Micro instance has one 32-bit or 64-bit virtual core, 615 MiB memory, and up to 2 *EC2 compute units*, for short periodic bursts. This last property requires some explanation. First of all, since an Amazon EC2 instance is a virtual server, the underlying physical processor can change over time. Nevertheless Amazon tries to provide a consistent amount of CPU capacity, so they define 1 *EC2 compute unit (ECU)* as the equivalent of one 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. The CPU capacity of each instance type is defined via ECUs, e.g. a general purpose large instance has 4 ECUs, but a compute optimized extra large instance has 88 ECUs. The Micro instance type is different from all the other instance types, because the CPU capacity we can use is variable, and the maximal 2 ECUs is only available for short bursts. As Fig. 1 illustrates, a Micro instance can operate at a low background level for longer periods, and can reach the maximum at short spikes. For this reason, Amazon recommends Micro instances for low traffic websites or blogs, small administrative applications and bastion hosts. Note that Fig. 1 does not have scale on either of the axes. This is due to the fact that Amazon does not specify how often a Micro instance is allowed to burst, but in case of too frequent or too long CPU bursts, the capacity of the instance is decreased significantly.

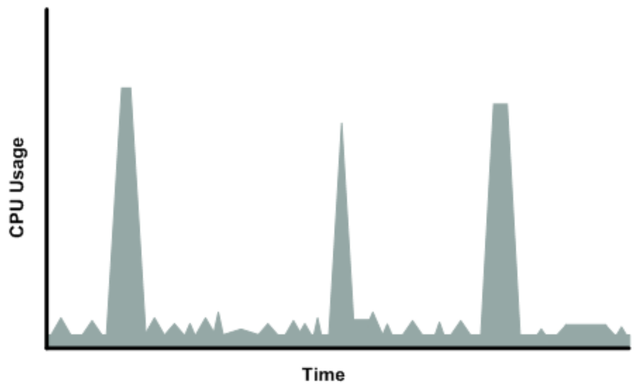

Fig. 1 The CPU capacity of Micro instances. Source: [13]

The other Amazon EC2 server we investigate is the general purpose Small instance. The Small instance can use 1.7 GiB memory, while the CPU capacity is constant 1 ECU, which means that we can use this maximum capacity without any restrictions. This capacity is provided by one virtual CPU core, that can be 32-bit or 64-bit, according to our choice. The Small instance is not for free, not even in the first trial year. Also, comparing the prices after the trial period, we find that the hourly price of a Small instance is currently three times as much as the hourly price of a Micro instance.

3. CONTRIBUTIONS

We investigated the scalability of the Apache Roller blogger web application deployed on an Amazon EC2 Micro and Small instances, via performance measurements. In this section, we describe the detailed setup of these measurements and discuss the results of them.

3.1. The measurement environment

The browser clients were emulated by Apache JMeter [14], an open source load tester tool. It was run on a PC with a 2.133GHz Intel(R) Core(TM)2 CPU 6600, 4 GB RAM, on top of 64 bit Microsoft Windows 7 Enterprise, with Service Pack 1, the Java Runtime Environment (JRE) was 64 bit Oracle JDK 6u45.

We executed the load tests against two server environments, which differed from each other only in the type of the EC2 instance: Micro and Small instances were testes. Both instances run in the EU region (Ireland) with the same software configuration. The operating system was Amazon Linux, we installed MySQL 5.5.31 Community Server as the database management system, 64 bit OpenJDK 6b12 as the Java Runtime Environment, and Apache Tomcat 7.0.40 as the web container needed to host the Roller web application.

3.2. The measurement methodology

We defined a test database in three different sizes ($DB1$, $DB2$, $DB3$), and defined three different loads of the clients (*Load*1, *Load*2, *Load*3). We executed the measurements in all possible database size-load combinations in both environments. Table 1 summarizes the three database sizes, while Table 2 describes the different client loads.

It is important to note that a larger database affects the performance in three ways:

1. The queries against a larger database can take more resources and time.

2. The response pages may contain more data (e.g. more comments per blog entry), so generating a page can take more time.

3. The size of the response page can be larger, causing a larger network time.

The different database sizes can correspond to different time intervals, e.g. the number of registered users doubles in a few months. Also, based on some blogs we follow, we assumed that a blog is used by more *Readers* than *Commenters*, and more *Commenters* than *Bloggers*.

The client loads are defined via the number of concurrent users with different behaviors, namely *Readers*, *Commenters* and *Bloggers*. A *Reader* user clicks an entry on the main page of a blog in a loop, next, he starts some search queries and clicks on some of the search results. To emulate realistic user behavior, some user think time is introduced between the requests, i.e. the user threads wait a normally distributed random time before sending the next request. The mean of this waiting time is 5 minutes for reading an entry, 1 minute for reading a search result, and 2 seconds for other requests.

Table 1 The definition of the test database sizes during the measurements

	$DB1$	$DB2$	$DB3$
Number of blogs	10	15	20
Number of entries in each blog	20	30	40
Number of comments for each entry	20	30	40
Number of registered users	100	150	200

Table 2 The definition of the client loads during the measurements

	*Load*1	*Load*2	*Load*3
Number of *Readers*	50	70	100
Number of *Commenters*	10	15	20
Number of *Bloggers*	5	7	10

A *Commenter* user picks an entry from a blog, then decides to comment it, so he logs in. Next, he starts a loop in which he sends a comment, than refreshes the page a couple of times so that he can read the answers of other commenters. The mean of the waiting times are 30 seconds for reading the entry, 1 minute for writing the comment, and 10 seconds before refreshing the page.

A *Blogger* logs in, starts to write a blog entry, than saves it as a draft. Next, he moderates the comments, i. e. reads the list of comments, and deletes some of them. Finally, he returns to the draft blog entry, finishes and publishes it. Writing the draft takes 1 minute, picking a comment to delete takes 30 seconds, and writing the final version of the entry takes 5 minutes.

We ran each test case (one load combined with one database size) for one hour, during which the *Readers*, *Commenters* and *Bloggers* all sent their requests concurrently to the server, and the average response time was measured by JMeter. The number of requests during an hour-long test case obviously depends on the client load level, but was typically in the order of a few thousand.

3.3. Memory issues on the Micro instance

As we mentioned in Section 2, an Amazon EC2 Micro instance has 615 MiB of memory. When we run our first tests, even at the smallest load and database size, we experienced that only after a few minutes of testing, the Tomcat service was shut down by the Linux OOM killer because of the lack of memory. It is worth to know, that Micro instances are by default configured without swap space, so that the number of I/O requests - that are billed after the first 2 million free requests per month - remains predictable.

Initially, we tried to handle the memory problem without adding swap space. We set a 256 MB limit for the Java heap size, but the Java Virtual Machine uses additional space beyond this, for shared libraries, class definitions, thread-local stacks, etc. Next we maximized the number of database connections at 10, and the number of Tomcat processing threads at 30, since both settings reduce the memory used by the system. This way, we were able to run some of the test cases with smaller loads and database sizes, but OOM killer was still activated at larger loads.

Finally, we decided to introduce 1 GB of swap space, out of which never was more than 300 MB used. In the following sections, we report the results with swap space enabled. It is interesting to note, however, that the results without swap space, where we were able to measure without OOM killer, did not differ significantly from the results with swap space enabled.

3.4. The measurement results

Fig. 2, Fig. 3, and Fig. 4 present the average response times measured on Amazon EC2 Micro, with *DB*1, *DB*2 and *DB*3 test database sizes, respectively. Note that because of the higher value in Fig. 4, we chose two different scales on the vertical axes. Similarly, subsequent figures will use either 1300 *ms* or 3200 *ms* as the maximal value on the vertical axis, according to the highest value to display.

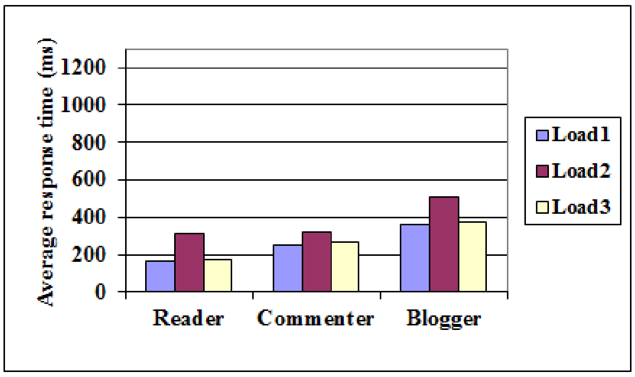

Fig. 2 Average response times on Amazon EC2 Micro with *DB*1

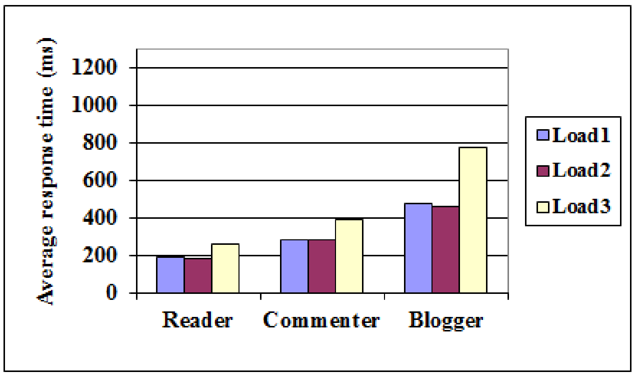

Fig. 3 Average response times on Amazon EC2 Micro with *DB*2

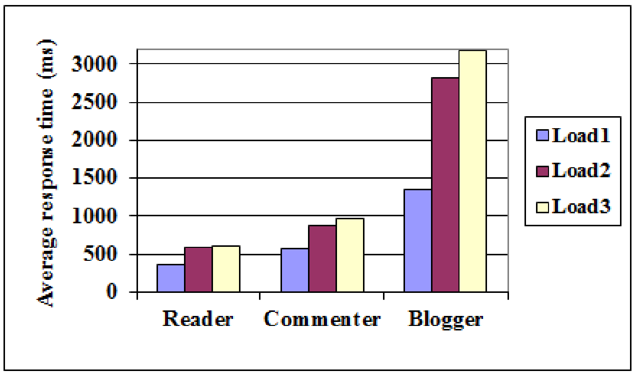

Fig. 4 Average response times on Amazon EC2 Micro with *DB*3

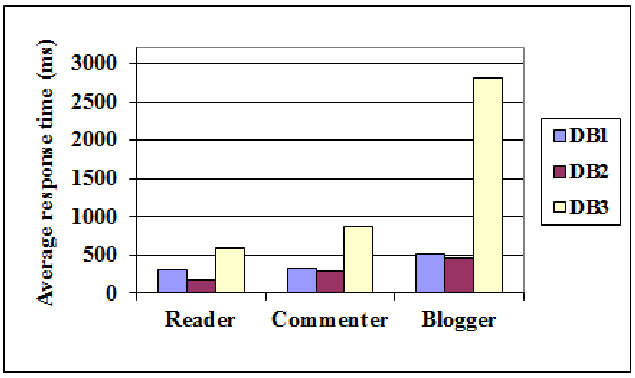

Fig. 5 Average response times on Amazon EC2 Micro with *Load*2

Investigating the diagrams, the conclusion is that in each test case, the *Bloggers* experience the highest response time, while *Readers* face the lowest one. According to our expectations, higher load and larger database would result higher response times, however, this is not fulfilled in some cases. In Fig. 2, it is conspicuous that the response times under *Load*2 are higher than under *Load*3. If we depict the response times with the load fixed at *Load*2, and growing database sizes in Fig. 5, we can find a similar result, that the response times in the $Load2 - DB1$ case are higher than in the $Load2 - DB2$ test case. The last (smaller) anomaly appears in Fig. 3, $Load2$ shows slightly better response times than $Load1$.

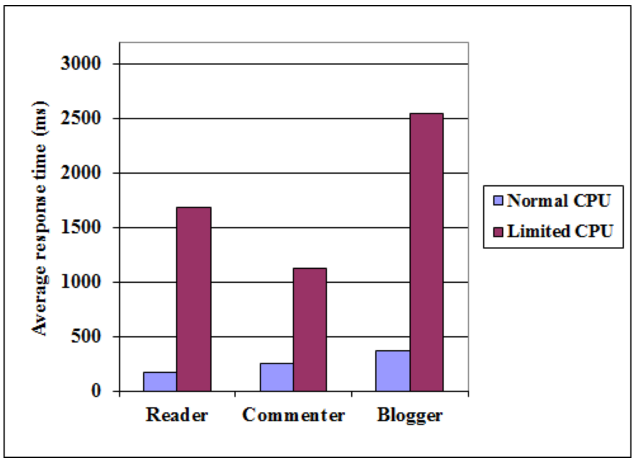

Fig. 6 The effect of CPU stealing ($Load1 - DB1$)

All these surprising results have the same explanation: the particular CPU capacity of the Amazon EC2 Micro instances. As mentioned in Section 2, Amazon periodically checks how much CPU resources the Micro instance has used. If this CPU usage is too high, it temporarily limits the available CPU capacity of the instance, i.e. it "steals" CPU cycles from our instance in favor of other instances running on the same physical CPU. The rate of the CPU stealing can be monitored using the Linux *top* utility, and can even exceed 90%. Obviously, such drastic CPU limitation causes a dramatic increase in response times, as Fig. 6 illustrates it. The second data series in the figure shows the result of a 3-minutes-long test run with $Load1$ and $DB1$, during which CPU steal rate was between 70% and 90%.

Amazon does not define exactly, in which periods the CPU usage is checked, what the allowed level of CPU usage is, and how long the CPU level is limited. It is likely that it varies with the capacity needed by other instances running on the same machine. For this reason, we cannot define an exact load, when the limitation of the CPU appears. During our measurements we experienced that if we start tens of JMeter threads in a short amount of time (e.g. 1-2 minutes), the CPU usage on the Micro instance jumps over 90%, and a drastic CPU limitation follows with a steal rate over 90%, which lasts several minutes. This made obvious what we expected, that an Amazon EC2 Micro instance cannot handle the *flash crowd* [15] effect. A flash crowd is a large spike in traffic to a website. A special case of flash crowd is the so called Slashdot effect, when the cause of the suddenly increased traffic is that a popular site with large traffic mentions or links to a smaller site. In our measurements we tried to eliminate this effect by defining a ramp-up period of 400 seconds, during which the client threads started gradually. Still, if later in the 60 minutes period of the measurement a congestion of the requests happens, CPU stealing might take place. These periods of limited CPU can be detected on the server side with the *top* utility, but the client side results reveal them as well, since JMeter registers the maximum response time for each request type. If this maximum exceeds 10 seconds, it is very likely that the CPU of the Micro instance was limited for a shorter or longer period during the measurement. The test cases that run without CPU limitation were $Load1 - DB1$, $Load1 - DB2$, $Load2 - DB2$, $Load3 - DB1$. This explains the earlier detailed counter-intuitive scaling behavior with respect to load and database size.

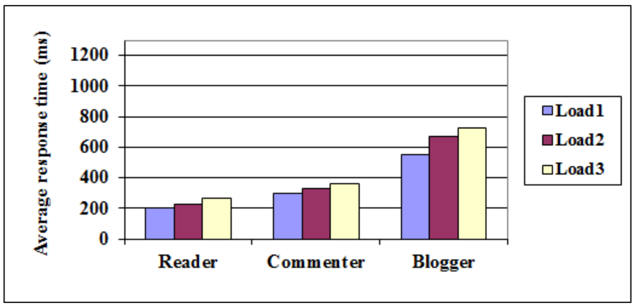

Fig. 7 Average response times on Amazon EC2 Small with $DB1$

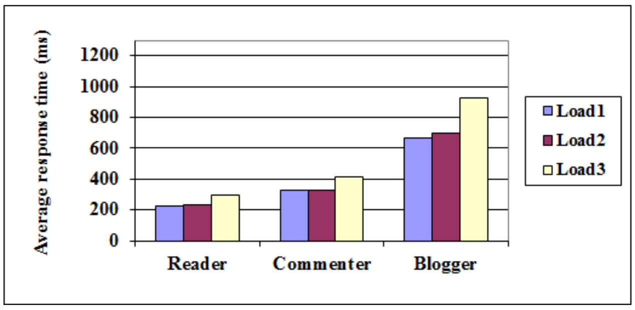

Fig. 8 Average response times on Amazon EC2 Small with $DB2$

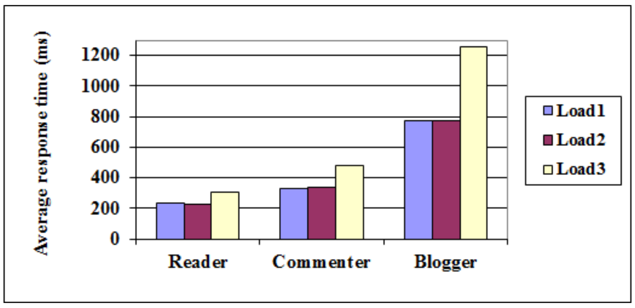

Fig. 9 Average response times on Amazon EC2 Small with $DB3$

The average response times measured with the Small EC2 instance are shown in Fig. 7, Fig. 8 and Fig. 9. Since Small instances have a consistent amount of CPU capacity (1 ECU), the surprising results observed at Micro instances do not occur. Consequently, larger load or larger database size cause a higher average response time in almost all cases. A few exceptions can be found at database sizes $DB2$ and $DB3$, where $Load2$ produces about the same results as $Load1$.

The most interesting conclusions are derived from the results of the Micro and Small instances depicted inFig. 10, Fig. 11 and Fig. 12. At $DB1$ and $DB2$ database sizes, the Micro instance almost always performs better than the Small instance. This can be explained with the fact that the maximum CPU capacity of the Micro instance is the double of the Small instance (2 versus 1 ECU). As long as the load is not so high that a significant CPU stealing occurs, our application can benefit from the high CPU spikes allowed. At $DB1$ and $DB2$ database sizes, we can find only one case, where the Small instance outperforms the Micro instance, namely the *Readers* in the $Load2 - DB1$ case experience a lower response time with the Small instance. In this particular case for the Micro instance, we have seen in Fig. 2 that CPU stealing caused higher response times.

Increasing the database size to $DB3$, Fig. 12 proves that the Small instance outperforms the Micro instance at all load levels. This means that the CPU requirement generated by these loads caused such amount of CPU stealing that the CPU spikes of 2 ECU cannot counterweigh the constant 1 ECU capacity of the Small instance.

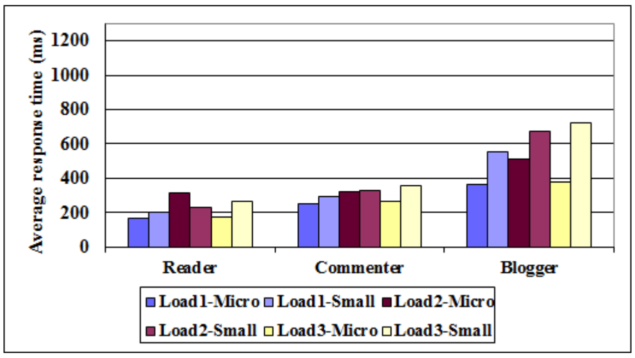

Fig. 10 Average response times on Amazon EC2 Micro and Small instances with $DB1$

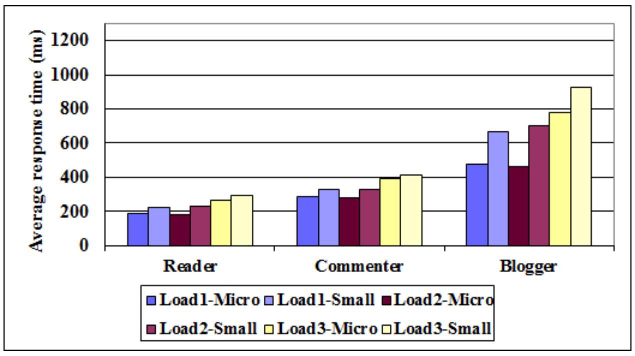

Fig. 11 Average response times on Amazon EC2 Micro and Small instances with $DB2$

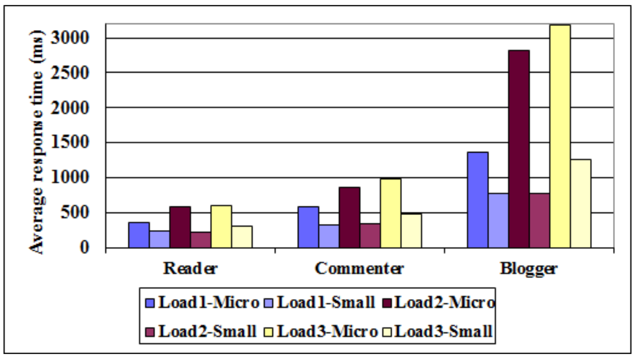

Fig. 12 Average response times on Amazon EC2 Micro and Small instances with $DB3$

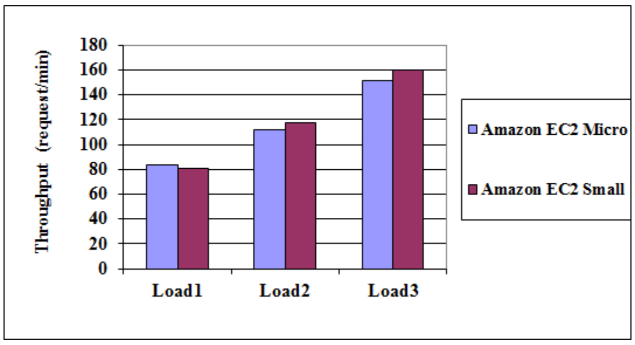

Fig. 13 Throughput of the servers with $DB3$

The throughput of the web application is shown in Fig. 13 for both the Micro and Small instances. We only display the case with the largest database size, since the test cases with the other two database sizes produced about the same results. We can observe that at the investigated load levels, the throughput values are nearly the same for both instance types.

4. RELATED WORK

Performance analysis of web applications has been a popular topic in the last decade. Several papers investigate different performance aspects of different applications. The paper of Nagpurkar et al. [16] is related to our work because it presents a detailed characterization of multiple selected Java web applications, and one of the applications is Lotus Connections. Lotus Connections is IBM's social software platform (nowadays called IBM Connections [17]), whose blog engine is built on Apache Roller. The behavior they define for testing the blogging feature of the application, is similar to ours: the users search, browse the blog entries, add comments, create new entries in a predefined mix. However, their focus is not the response time perceived by the clients, but the server-side workload, so they mainly analyze the hardware level performance metrics like CPU utilization, cache misses, stalls, etc. They conclude that Web 2.0 workloads cause different server-side behavior opposed to traditional OLTP workloads.

Although not dealing with performance analysis, [18] still relates to our work. The authors propose a formal framework for managing the evolution of an application originally designed as a single-tenant system for on-premise usage to a multi-tenant SaaS system. Their approach is illustrated with the Apache Roller application. So if someone plans to run a multi-tenant blog engine as a SaaS, the proposal of [18] can be used to transform Apache Roller, and our paper gives some insight into the potential performance of the blog engine when deploying it on Amazon EC2 Micro or Small instances.

More recently, the performance of applications that are deployed in the cloud are of growing interest. A large part of the related research focuses on scientific applications with high performance need. Ostermann et al. [19] conclude that the performance and the reliability of Amazon EC2 is insufficient for scientific computing at large. Similar statements are made in [20], where the authors found that using representative application workloads, Amazon EC2 is six times slower than a typical mid-range Linux cluster, and twenty times slower than a modern High Performance Computing system. They also revealed a strong correlation between the percentage of time an application spends communicating, and its overall performance on EC2. These results are consonant with those published in [21], stating that smaller, too frequent communication between nodes spoil the overall performance of an Amazon EC2 cluster, but less frequent, large data exchanges are acceptable.

The authors of [22] found that the performance of Small instances is relatively stable and the mean response time remains within 8% of the long-time average. However, different Small instances, which should provide identical performance, can perform very differently up to a ratio 4. During our experiments, we did not experience such a variance in the performance of the Small instances.

A detailed performance and scalability analysis of an n-tier web application deployed on different cloud platforms is done in [23]. The authors investigated RUBBoS [24], an n-tier electronic commerce system, on three different cloud platforms: Amazon EC2, Open Cirrus, and Emulab [25]. They tested several deployment configurations with respect to the number of web servers, Java web containers, and numbers of SQL and data nodes in the MySQL cluster, thus testing horizontal scalability. In the case of Amazon EC2, vertical scalability was examined as well, by changing the instance types (small, large, extra large, and cluster). The results show a good vertical scaling for Amazon EC2, and good horizontal scalability on Open Cirrus and Emulab. The horizontal scalability of Amazon EC2 was, however, very poor: increasing the number of MySQL data nodes caused a degradation in the throughput, although the CPU utilization of the servers was not saturated. The authors found the cause of this behavior in the large number of client threads, larger context switching time, and network latency.

5. CONCLUSIONS

We have investigated the performance of an open source Java weblog engine running on different server types in the cloud. We have deployed the application in the Amazon EC2 cloud, on Micro and Small instances. The Small instance provides a consistent CPU capacity of 1 ECU, while the Micro instance allows CPU spikes of 2 ECU, but throttles the CPU capacity at a very low level, if CPU spikes are too frequent.

We have sent requests to both server types in the name of concurrent users using a load tester tool. We have examined three different database sizes and three different client loads. We have found that at the largest examined database size, all of the examined client loads could be served faster with a Small instance, due to the more intensive CPU stealing at the Micro instance. At lower database sizes, however, the Micro instance proved to perform better.

Thus we can conclude that the choice between Amazon EC2 Micro and Small instances should consider the load of the instances, since lower load levels can be served not only cheaper, but also with better performance using Micro instances. Our results also suggest that an adaptive load balancer, dispatching only that much amount of requests to Micro instances so that CPU throttling is avoided, can lead to a more cost effective operation of web applications in the Amazon Elastic Compute Cloud. Our future work will focus on developing a prototype for such a load balancer.

Another possible direction for future work is to elaborate an analytical model that can calculate the load levels at which an application should be deployed on Micro or Small instances. Such model has to take the hourly prices of the instance types and the service time distribution of the requests into account.

ACKNOWLEDGEMENT

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) organized by VIKING Zrt. Balatonfüred. This work was partially supported by the Hungarian Government, managed by the National Development Agency, and financed by the Research and Technology Innovation Fund (grant no.: KMR_12-1-2012-0441).

REFERENCES

[1] Future of Cloud Computing, North-Bridge, 2012, `http://northbridge.com/2012-cloud-computing-survey/`

[2] The NIST Definition of Cloud Computing, National Institute of Standards and Technology, 2011, `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`

[3] Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 4Q12, Gartner, 2013.

[4] Survey Analysis: CFOs' Top Imperatives From the 2013 Gartner FEI CFO Technology Study, Gartner, 2013.

[5] ARMBRUST, M. et al.: *A view of cloud computing*, Communications of the ACM **53**, No. 4 (2010) 50–58

[6] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`

[7] IMRE, G. et al.: Performance of a Java web application running on Amazon EC2 Micro instance, in 12th International Conference on Informatics, 2013, 204–209

[8] Apache Roller blog engine. `http://roller.apache.org/`

[9] Apache Lucene search engine. `http://lucene.apache.org/`

[10] JRoller Java blog. `http://www.jroller.com/`

[11] ZHANG, Q. et al.: *Cloud computing: state-of-the-art and research challenges*, Journal of Internet Services and Applications **1**, No. 1 (2010) 7–18

[12] Amazon EC2 instance types. `http://aws.amazon.com/ec2/instance-types/`

[13] Specification of Amazon EC2 Micro instances. `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts_micro_instances.html`

[14] Apache JMeter load tester. `http://jmeter.apache.org/`

[15] NIVEN, L.: The flight of the horse. Ballantine Books, 1973, ISBN 0-3452-3487-1

[16] NAGPURKAR, P. et al.: Workload characterization of selected JEE-based Web 2.0 applications, in IEEE International Symposium on Workload Characterization (IISWC) , IEEE, 2008, 109–118

[17] Home page of IBM Connections. `http://www-03.ibm.com/software/products/us/en/conn`

[18] JU, L. et al.: Tenant onboarding in evolving multitenant Software-as-a-Service systems, in IEEE 19th International Conference on Web Services (ICWS), IEEE, 2012, 415–422

[19] OSTERMANN, S. et al.: A performance analysis of EC2 cloud computing services for scientific computing, Cloud Computing, Springer, 2010, 115–131

[20] JACKSON, K. R. et al.: Performance analysis of high performance computing applications on the amazon web services cloud, in IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2010, 159–168

[21] HILL, Z. et al.: A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?, in 10th IEEE/ACM International Conference on Grid Computing, IEEE, 2009, 26–33

[22] DEJUN, J. et al.: EC2 performance analysis for resource provisioning of service-oriented applications, in Int'l Conference on Service Oriented Computing. ICSOC/ServiceWave 2009, Springer, 2010, 197–207

[23] JAYASINGHE, D. et al.: Variations in performance and scalability when migrating n-tier applications to different clouds, in IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2011, 73–80

[24] RUBBoS: Bulletin board benchmark. `http://jmob.objectweb.org/rubbos.html`

[25] Emulab - Network emulation testbed. `http://www.emulab.net`

Received November 18, 2013, accepted Dec 17, 2013

BIOGRAPHY

Gábor Imre graduated (M.Sc. in Software Engineering) with distinction in 2004. He is an Assistant Lecturer in the Department of Automation and Applied Informatics at the Budapest University of Technology and Economics. His research fields focus on software engineering, performance analysis, performance modeling, cloud computing and Java Enterprise Edition. He won the IBM Faculty Award (2009) and the IBM Ph.D. Fellowship Award (2006).

Hassan Charaf received his PhD in 1998. He is an Associate Professor and fellow in the Department of Automation and Applied Informatics at the Budapest University of Technology and Economics. He is the head of the IT group. As an out-standing figure in teaching, research and development he is in key positions at several organizations at the university. His research fields are: software modeling and model processing; mobile platforms; distributed systems and cloud computing; and data technologies.

László Lengyel received his PhD in 2006. He is an Associate Professor and fellow in the Department of Automation and Applied Informatics at the Budapest University of Technology and Economics. His various research fields focus on software modeling, metamodeling, graph rewriting-based software model transformation, model-driven development, constraint validation, validated model transformation and aspect-oriented techniques. The most important milestones in his professional career include, but are not limited to: the Bolyai János professorship (2006-2010), the Siemens Excellence Award (2008), and being chosen as the recipient of the NJSZT Kemény János-award (2012).