Performance Testing Web Applications on the Cloud

Conference Paper · March 2014

DOI: 10.1109/ICSTW.2014.57

CITATIONS
11

READS
696

3 authors, including:

Joydeep Mukherjee
York University
11 PUBLICATIONS **69** CITATIONS

SEE PROFILE

Performance Testing Web Applications on the Cloud

Joydeep Mukherjee
ECE Department
University of Calgary
jmukherj@ucalgary.ca

Mea Wang
CS Department
University of Calgary
meawang@ucalgary.ca

Diwakar Krishnamurthy
ECE Department
University of Calgary
dkrishna@ucalgary.ca

***Abstract*—Web applications are increasingly resorting to public cloud platforms such as the Amazon Web Services (AWS) Elastic Compute Cloud (EC2). However, previous studies have shown that the virtualized infrastructures used in a cloud environment can introduce performance issues. Hence, it is crucial to test the performance of the cloud to support Web applications. This is challenging because the performance effect of the cloud platform cannot be easily isolated from other extraneous factors. In this paper, we present a systematic experimental process to address this challenge. Following our proposed process, we test the performance of Web applications running on different types of AWS EC2 instances. Results suggest that Web server response times can fluctuate up to 1000% for the same workload under certain circumstances such as instance type, time-of-the-day, and day-of-the-week.**

I. INTRODUCTION

Web applications are now moving to the cloud platform since it has many advantages over conventional bare-metal data centers. Large businesses can benefit economically from using virtualized cloud services instead of buying and managing expensive hardware. One of the key reasons for this move is the great scalability of the cloud, empowered by the virtual machine (VM) technology. Cloud users can add, remove and migrate VM cloud instances easily, and the process takes very little time. In Web applications users are interested in immediate response and are generally waiting for the requests to be executed before moving to other activities. Performance degradation in Web applications due to long request response times can cause customer dissatisfaction leading to financial losses for businesses operating these Web sites. For this reason it is important to choose a cloud platform that offers good and consistent performance.

Many studies [1], [2] have suggested that virtualized infrastructures, the base of typical cloud systems, can cause performance problems, *e.g.*, virtualization overhead and resource time-sharing. Furthermore, performance degradation can be introduced as a result of cloud-based management activities such as VM migration. Some of these actions can have unpredictable effects on the performance of Web applications running in the cloud. Performance of a VM depends to a large extent on what other VMs running on the same physical host are doing. To better understand the performance issue, studies [3], [4] have been conducted to examine the performance of batch applications running on a cloud platform, but no study has been done for interactive Web applications.

In this paper, we consider the popular Amazon Web Services (AWS) as the cloud platform. AWS offers cloud services such as the Elastic Compute Cloud (EC2), which is commonly used to host many popular Web services such as Netflix. EC2 offers different types of instances using the open-source Xen virtualization middleware [5]. The main goal of this study is to examine the performance of Web applications running on the EC2 platform. Towards this objective, we focus on studying whether EC2 delivers consistent performance to support Web applications.

Surprisingly, isolating the performance impact of EC2 is very challenging. It needs a very carefully thought out experimental process. In this paper, we suggest a process for testing the performance of Web applications on the cloud that reduces the likelihood of making erroneous conclusions about the performance impact of the EC2 platform. More specifically, we present systematic testing procedures for selecting the location of the load generator used to generate test workloads, for choosing the right type of workload generator, for validating whether the network bandwidth available in the test setup is sufficient and stable, and for verifying the properties of the workload generated by the workload generator.

Following our testing procedure, we select three different EC2 instance types- small, medium and large. We study their performance when they are subjected to varying intensities of load. Our goal is to analyze the performance of these instances when running a Web application under similar load and to understand the performance variability and scalability of these instances. Since the performance of Web applications needs to be consistent over time, we monitor each instance every hour of the day for one week.

According to the results obtained from this experiment, the performance delivered by the EC2 platform seems to be quite unpredictable when running Web applications. We observe wide variations in performance of all three instances over the period of a week. A small instance can have more than 1000% increase in server response time over a period of 24 hours. Small and medium instances were observed to have over 100% increase in server response times for three days of a week. While a large instance performs better than the small and the medium in terms of response time and throughput, our results show that the response times from a large instance can vary as much as by 30% depending on the time of the day and the day of the week. These results suggest that organizations moving their Web services to the cloud should put in place mechanisms to mitigate potential risks from such performance

TABLE I
AMAZON EC2 INSTANCE TYPES

Instance Type	CPU Units	Cores	Memory [GB]	Disk [GB]	Cost [$/h]
Small	1	1	1.7	160	0.007
Medium	2	1	3.7	320	0.013
Large	4	2	7.5	850	0.026

variability.

II. TEST PROCEDURE

For the purpose of this work, we used three types of Amazon EC2 instances: small, medium and large. The characteristics of these instances are summarized in Table I. The amount of CPU that is allocated to a particular instance is expressed in terms of CPU Units in the table. One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 AMD Opteron or 2007 Intel Xeon processor. The number of cores refer to the number of virtual CPUs (VCPUs) in the instance. All these instances are located in the same region (US-West-Oregon) and their IP addresses belong to the same network subnet. Ubuntu server version 12.04 (kernel version 3.2.0-40-virtual) is installed as the Operating System in all three instances. We also installed Apache 2 (version 2.2) as the Web server, PHP with FastCGI (version 5.3.6) as the application tier, and MySQL (version 5.1.49-1) as the database server in all instances. Collectl [6] is used in each instance to monitor the CPU utilization.

Performance degradation arising from complexity inherent in multi-tier Web applications can be difficult to isolate from performance issues in the cloud due to factors such as virtualization overheads. To better isolate the performance impact of the EC2 platform, we by design chose a simple workload for this study. In this workload, each request invokes a PHP script on the Web server that incurs an exponentially distributed CPU service demand, *i.e.*, CPU time of 0.02 milliseconds.

We use the *httperf* Web workload generator tool [7] to generate synthetic requests to the Web Server. *httperf* submits requests to a Web application by establishing multiple concurrent connections with the application and reports metrics such as end user response times and application throughput. It can be configured to vary factors such as the load, *i.e.*, connections per second (cps) and the number of requests sent to the server. Tests were conducted by submitting workloads through *httperf* to the applications hosted on each of the three instances. In our study, *httperf* is executed on a host or an EC2 instance other than the server instances. Further details regarding the experiment setup are as follows. The time between successive connection arrivals, i.e., inter-arrival time, is chosen to be exponentially distributed. Every *httperf* test to an instance runs for a period of 100 seconds. To obtain statistical confidence in mean response times, multiple runs are conducted in each experiment and a 95% confidence interval (CI) is calculated for the mean response time. For all experiments reported, these confidence intervals are within 5% of their corresponding mean values.

We conduct a subset of experiments over a week to look for performance variability in the EC2 platform. In these experiments, *httperf* is run at the beginning of each hour for 24 hours in a day over the course of a week. Each *httperf* test to an EC2 instance runs for 100 seconds as mentioned before. Performance data from the instances was captured only once every hour of the day for a week in order to minimize the cost of running the EC2 instances continuously.

We use response times obtained from *httperf* to infer the response times of the server under testing. We note that although end to end response time is an important metric, we only focus on server response time in our study. As a result, factors such as Internet latency which can impact end to end response time are ignored. Response time of a request is composed of two parts: connect time and reply time. Connect time is the time taken to establish a TCP connection with the server. Reply time is defined as the time spent from sending the first byte of a request until receiving the last byte of the reply on an already established connection. It is important to note that response times obtained from *httperf* are a good indicator of the test Web server's response time only when there are no bottlenecks in the host or EC2 instance executing *httperf* and the network supporting the test setup. Consequently, care must be exercised to rule out such bottlenecks in the test setup, which will be detailed in the testing procedure presented in this section.

Our test procedure is as follows. We first run experiments to rule out bottlenecks in the test setup that can prevent us from inferring the true impact of the EC2 platform on Web server response time. Specifically, we conduct experiments for determining whether the workload generator should be located inside the cloud or in a dedicated physical host in a remote location. Based on these results, it was clear that the performance impact of the EC2 platform is better evaluated when the workload generator resides inside the cloud. We next conduct experiments to determine the right type of EC2 instance to serve as the workload generator. The experimental results show how choosing a wrong instance as the workload generator can badly affect our estimates of server performance. Next, we continuously monitor the network bandwidth between the workload generator and the Web server instances under test using the Iperf tool [8]. After monitoring the network for a week, we analyze the results to see if there is a wide variation in available bandwidth and whether the available bandwidth is sufficient to sustain our test workloads. Finally, we run experiments to analyze the performance of the three types of EC2 Web server instances.

III. VALIDATING THE TEST PROCEDURE

In this Section, we present experiments to validate our test procedure for testing the performance of Web applications on the EC2 platform.

A. Using a Remote Workload Generator

Results obtained from testing the performance of a cloud platform can vary depending on the location of the workload generator. A remote workload generator located in our data center will give us control over the workload generator host. For example, we can ensure that no other programs are running on the workload generator host when running our tests thereby eliminating a source of workload generator host bottleneck. However in this case, each request to and from the EC2 Web server instance has to travel over the Internet. As a result, response time data collected from *httperf* will include the uncontrollable variations induced by the Internet traffic. On the other hand, placing the workload generator inside EC2 might be problematic. EC2 instances are virtualized and hence the performance of a *httperf* instance might be compromised by that of other EC2 instances sharing the same physical host. This can in turn introduce workload generator bottlenecks that render *httperf*'s reported response times to be not a good indicator of the response time of the EC2 Web server instances. Furthermore, the available network bandwidth between the workload generator instance and the server instance may not be sufficient or stable to sustain the test workloads, thereby making it unreliable to use *httperf* reported response times.

We present a simple experiment to show the effect of the location of the workload generator when testing the performance of the cloud platform. The workload generator was chosen to be a host (2 CPU, 4 cores per CPU, 4 x 10 Gbps NIC, AMD Shanghai Opteron 2378 server) in a data center in University of Calgary. We refer to this setup as the *initial setup*. Table II shows the performance where R represents the average reply time of a request in milliseconds and C represents the average connect time of a request in milliseconds of each instance when the workload generator is situated outside the cloud.

According to Table II, Internet round trip latencies dominate the response times measured by *httperf*. For example, the average connect time of a request sent to each instance is quite high. Similar tests performed in our lab where the server and the workload generator are connected directly to each other using a dedicated 1 Gbps network connection show that the average connection time of each request is generally in the order of 0.1 to 0.2 milliseconds. The lowest average connect time per request value at even low cps values (for e.g. 100 cps) in the *initial setup* is 25.2 milliseconds. Also, there is very little difference between the reply times of the instances to suggest any performance improvement when we go from a small to a large instance. This is because the cores of all three instances were very lightly loaded in these tests and the reply time is dominated by the Internet round trip latency. It should be noted that the maximum throughput of a small, medium and large instance is 300 cps, 600 cps and 1200 cps respectively. Beyond these cps values, we get many connection timeout errors from *httperf* which indicates that the server instances have reached their maximum processing capabilities. For all the cps values shown in Table II, the connect time is almost 50% of the total response time, thus indicating that the total server response time is dominated by the round-trip latency incurred over the Internet. These results show how the influence of the Internet makes it difficult to infer the actual performance of the instances inside the cloud platform when we use a remote workload generator.

TABLE II
PERFORMANCE USING THE INITIAL SETUP

CPS	Small-R	Small-C	Med-R	Med-C	Large-R	Large-C
100	26.9	25.7	26.3	25.3	26.9	25.2
200	26.9	25.5	26.5	25.2	27.1	25.3
300	28.9	27.2	27.0	25.2	29.2	27.4
600			29.5	25.2	29.6	28.4
1000					32.1	33.4
1200					34.5	33.8

B. Using In-Cloud Workload Generator

In order to avoid the confounding effect of the Internet, we decided to create another instance in the cloud along with the other server instances to serve as the workload generator. We refer to this setup as the *final setup*. This decision was taken in light of the knowledge that the workload generator instance can be created in the same region (US-West-Oregon) as the rest of the instances and the IP address of the workload generator instance will share the same network subnet with the server instances.

The next challenge is to determine the type of instance to serve as the workload generator inside the cloud. As seen in Table I, the cost and resource capabilities of the instances increase progressively as we go from small to large instance. Keeping this in mind, we need to choose the right type of instance for the workload generator so as to get reliable performance data when testing the server instances. With this objective in mind, an experiment was conducted where the performance of each server instance was measured by using a small, a medium and a large instance each as a workload generator in our *final setup*. The results are shown in Table III. The performance of each EC2 instance in Table III is measured in terms of response time (in milliseconds). The average connect time for a request for all three types of workload generators is in the order of 0.1 to 0.2 milliseconds even at high cps values, thus validating our decision to move the workload generator host inside the cloud. It was also observed that the response times obtained from all three instances were much lower than the ones seen in our *initial setup* in Table II.

Results from Table III show the importance of choosing an appropriately sized workload generator instance. Using the wrong type of workload generator instance can provide misleading insights into the performance of the Web server instance under test. For example, consider the use of a small workload generator instance and a large workload generator instance to test the performance of a large server instance at 1200 cps. From Table III, the small *httperf* instance reports a response time that is almost 3 times that reported by the large

TABLE III
PERFORMANCE OF DIFFERENT WORKLOAD GENERATORS

Workload Generator	CPS	Small Instance	Medium Instance	Large Instance
small	100	14.7	11.5	5.9
	200	22.0	11.8	6.6
	300	100.7	18.0	7.1
	600		89.4	15.6
	1000			30.1
	1200			36.2
medium	100	12.1	7.6	4.5
	200	19.6	10.3	4.7
	300	93.4	15.2	5.6
	600		80.1	10.1
	1000			19.2
	1200			28.4
One large	100	10.9	5.2	3.6
	200	17.2	8.2	3.8
	300	87.6	12.3	3.9
	600		73.3	5.0
	1000			7.3
	1200			12.1
Two large	500			7.4
	600			12.3

httperf instance. Using the small *httperf* instance will lead the tester to be incorrectly pessimistic about the response time of the server instance. Essentially, the tester will be confounding a workload generator instance bottleneck to be a performance problem at the server.

Results from Table III suggest that a large workload generator instance is needed for testing all three types of Web server instances. For choosing the correct workload generator in any cloud environment, tests must be performed by progressively choosing bigger instances to serve as the workload generator until the *httperf* response times converge. As seen from Table III, the response times progressively decrease as we move from a small workload generator instance to a large workload generator instance regardless of the type of server instance being tested. It should also be noted from the table that when two large instances are used as the workload generators with each instance sending requests at 500 and 600 cps, the response times observed are nearly the same as obtained from one large instance sending requests at 1000 and 1200 cps, respectively. This experiment demonstrates that there is no significant impact on *httperf* reported response times because of any workload generator side bottlenecks when using a large instance. From these results, we decide to choose a large instance as our workload generator in the rest of the paper.

C. Measuring Network bandwidth for Tests

Large variations in the network bandwidth available between the workload generator and the server can account for huge variations in how *httperf* perceives server performance.

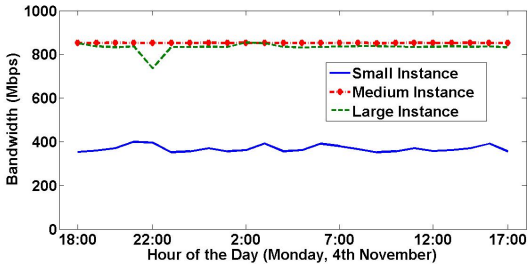

Fig. 1. Hourly available bandwidth

In order to ensure that the available bandwidth is constant and sufficient for testing our workload, we monitored the available network bandwidth between the workload generator and the server instances every hour for 24 hours using the Iperf tool. We ran our test on 4th November, Monday, starting at 18:00 and collected hourly available bandwidth data by running tests at the beginning of every hour for the next 24 hours.

The results from this test are shown in Figure 1. The available bandwidth between the workload generator and the small instance shows some variations for different hours of the day. The bandwidth varies between 352 Mbps and 399 Mbps with a mean of 368 Mbps. However, the Coefficient of Variation (COV), *i.e.*, the ratio of standard deviation to mean, is only 0.04, which suggests that the bandwidth can be considered as practically stable throughout the day. The available bandwidth for the medium and large instances are also stable throughout the day with a very low COV value, a drop in bandwidth for the large instance at the fifth hour notwithstanding. Interestingly, the available bandwidth for a medium instance in a day is slightly higher than a large instance. Moreover, the average network bandwidth for a medium and a large instance is more than twice the average network bandwidth available for a small instance. It was also noticed from the *httperf* output files that the average network bandwidth corresponding to the maximum loads for a small, medium and large server instance are 134 Kbps, 162 Kbps and 188 Kbps, respectively. This indicates that the available bandwidth is sufficient for running our tests. In order to further confirm the stability of the network, the network was monitored for one week on a per-hour basis, *i.e.* by running the test at the beginning of every hour. It must be noted that the trend of the graph shown in Figure 1 was observed consistently throughout the entire week starting from 4th November, 2013 to 11th November, 2013. The results indicate that the network is stable and sufficient network bandwidth is available at all times for testing the performance of the server instances.

As a final step, we verify that *httperf* generates a workload with properties specified as its input. From the results explained in Section III-B, we chose a large instance as the host for the workload generator in our *final setup* for testing the server instances on the cloud. The test output log files from the *httperf* tool contains the timestamp of each request generated. We first verified from the traces that the total duration of the test is 100 seconds, *i.e.* the time we specified in the *httperf*

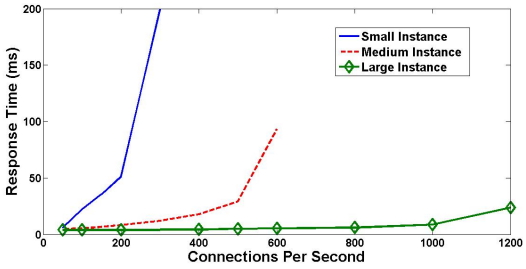

Fig. 2. Performance of EC2 instances

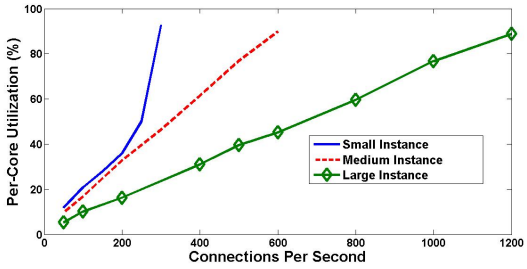

Fig. 3. Per-core Utilization of EC2 instances

command line. We also verified that the mean of the inter-arrival time between the requests is also exactly the same as that has been specified in the *httperf* command line. We then ran the Chi-Square goodness of fit test to verify that the inter-arrival time of requests generated by the workload generator follows an exponential distribution with the same mean value as specified in the *httperf* command line. We also analyzed the Q-Q plot between the empirical quantile data of the inter-arrival time from the *httperf* output logs and the quantile data from an expected exponential distribution generated from our data. The Q-Q plot shows that the inter-arrival time between the requests in our workload is indeed exponentially distributed as specified.

IV. Performance Results from EC2

After following the procedure suggested in Section III, we started testing the performance of the three Web server instances in the EC2 platform. Figure 2 shows the performance of the three instances under varying amounts of load. As seen from the figure, performance of the small instance is the worst whereas performance of the large instance is the best both in terms of the response time and the throughput. In terms of throughput, a large instance can support four times as much throughput (1200 cps) as a small instance (300 cps) can, and twice as much as a medium instance (600 cps) can. Beyond this throughput, the mean connect time of a request increases and we get many connection timeout errors from *httperf* similar to what we got for Table II, indicating that the server has reached its maximum processing capability. The per-core utilization at this point is also very high, which explains the high response time value. Figure 3 shows the per-core utilization of the three instances under varying amounts of load. As seen from the figure, all three instances incur a high per-core CPU utilization (in the range of 90%) at high arrival rates. In terms of response time, as seen from Figure 2, the response time of the small instance is the highest and the large instance is the lowest at the same cps. For example, at 200 cps the response time of the small instance is about 17 times higher than that of the large instance.

The next set of experiments is designed to check the consistency of the performance of a Web application over a longer period of time. We test the hourly performance of the three types of EC2 instances for a week to observe any performance variability in the EC2 platform. For this purpose,

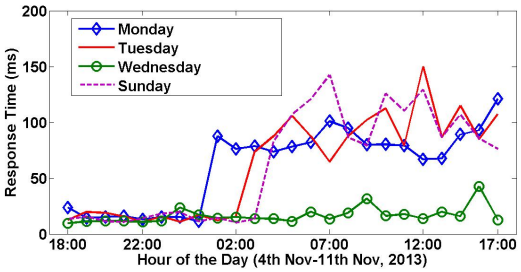

Fig. 4. Weekly performance of a small instance

we collected hourly performance of the three instances at a cps when around 50% of each core of the instance is utilized. We chose a low utilization level since operators typically underutilize servers in order to take into account the possibility of sudden workload bursts. From Figure 3, the small instance is 50% utilized at 250 cps, the medium at 300 cps and the large at 700 cps. We run the test for each instance starting at 18:00 every day for one complete week starting from 4th November, 2013 until 11th November, 2013. We also recorded the hourly available network bandwidth between the workload generator and the instances for a week. The bandwidth is stable throughout the day for every day in the week, similar to what is shown in Figure 1.

Figures 4, 5 and 6 show the performance trends of the small, medium and large instances over the week. Let us consider Figure 4, which shows the performance of a small instance recorded over a week. The response time of the small instance on Wednesday is indicative of its performance on Wednesday, Thursday and Friday, when the response times are comparatively low and stable (there are two hourly recordings where there is an increase of around 10%-15% over the mean response time recorded for the day on Wednesday). But for three days in the week- Monday, Tuesday and Sunday- we observe an increase of more than 1000% in response time after the first few hours. The response time can vary from a minimum of 10.1 ms (recorded on Thursday) to a maximum of 149.3 ms (recorded on Tuesday) over the duration of a week. Wide variation in performance is also observed on a single day, *e.g.*, on Tuesday we see an increase of over 1000% in response time. These observations lead us to conclude that the performance of a small instance hosting Web applications can be highly variable depending on the time of the day and

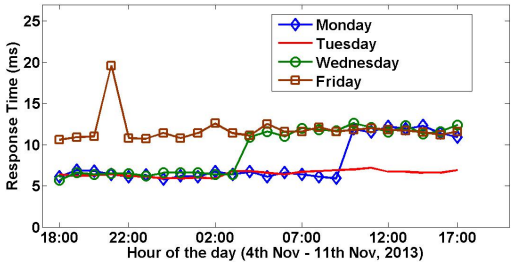

Fig. 5. Weekly performance of a medium instance

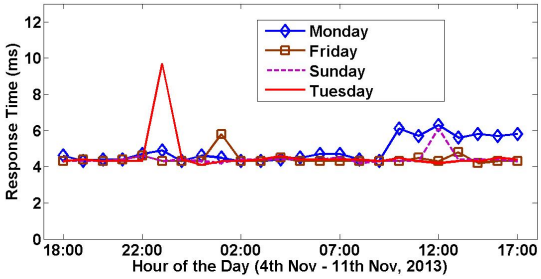

Fig. 6. Weekly performance of a large instance

the day of the week.

Figure 5 shows the weekly performance of a medium instance. From this figure, we observe variations in performance on three different days of the week: Monday, Wednesday and Friday. The performance on Tuesday is indicative of the performance for the rest of the week where there is not much variation in the hourly response time values. In the worst case scenario, on Monday, there is an increase of around 100% in response time after the first few hours. The average response time of the medium instance on Friday is also around 100% higher when compared to the average response time on a day on which the COV between the hourly response time values was low, *e.g.*, on Tuesday. All these observations point out the fact that the performance of a medium instance depends on the time of the day and the day of the week, similar to what was noticed earlier in case of a small instance.

Finally, Figure 6 shows the weekly performance of a large instance. The performance plot on Tuesday is representative of the performance for days in the rest of the week. The performance of this instance is relatively stable when compared to the performances of a small and a medium instance. However, there are many peaks in the response time plots on multiple days where there is more than 30% increase in response time, *e.g.*, on Friday and Monday. For some peak values, *e.g.*, on Tuesday, the response time is almost doubled at 23:00 as compared to the average response time throughout the day. This suggests that although the performance of a large instance is stabler than the small and the medium instances, performance can still vary depending on the time of the day and the day of the week.

From these results, it appears that the response time of a Web application running on the EC2 platform can be quite unpredictable. We have observed large variations in hourly response time from small and medium instances. While the response time of a large instance is relatively stable compared to the other two, it also has considerable variability depending on the time of the day and the day of the week. In our study, the server instances were run at a cps that utilizes 50% of each core in the instance. At higher per-core utilizations, the performance variation problem is likely to be more severe. Organizations that are looking to do capacity planning for hosting Web applications on the cloud should be aware of this sort of performance inconsistency over a long period of time and take actions to mitigate risks arising from such kind of behaviour. From our results, it appears that EC2 features such as Auto Scaling that automatically provision more resources in case of bad performance might benefit from using response time as an indicator of performance degradation rather than per-core utilization. Auto scaling reduces the need to provision Amazon EC2 capacity in advance. For example, it is possible to add new EC2 instances when the average CPU utilization of the existing EC2 instances goes above a specified threshold. However, our results indicate that increasing the number of EC2 instances automatically through Auto scaling based on utilization alone can be inefficient.

V. Background and Related Work

Performance testing of Web applications inside a cloud can be challenging. Previous works [9], [10] have discussed the issues, challenges, needs and practices involved with testing cloud environment. However, in contrast to our work, they do not focus on Web applications and do not demonstrate experiments on a real cloud environment. Previous work such as [11] that focused on benchmarking Web based applications on the cloud have not considered the challenges involved with testing the cloud environment to isolate the server performance from external factors.

Previous studies have focused on performance scalability and variability for batch and scientific applications hosted on the cloud. Kiruthika *et al.* [12] compared the performance of the three types of Amazon EC2 instances and proposed an algorithm for managing resource allocation in the cloud environment. A similar work was done by Ostermann *et al.* to analyze cloud computing services for scientific computing [4]. In-depth performance analysis of scientific workloads running on Amazon EC2 instances were done. At the end of the study, suggestions were made for improving the performance of cloud for scientific computing. Iosup *et al.* studied the performance of Amazon AWS and Google App Engine (GAE) platforms for batch type workloads over a period of two months [3]. They concluded that the performance exhibit time-dependent characteristics and show daily and monthly pattern. The traces for this study was collected from the CloudHarmony Web service [13], which records the performance of various EC2 instances over long periods of time for different micro-benchmarks including the LAMP Web server benchmark. However, the LAMP benchmark only provides a score based on maximum throughput and does not consider

response time. Moreover, it does not appear that these tests followed the process we did to ensure the throughput metric truly captures the performance of the cloud platform.

VI. CONCLUSION

In this paper, we highlighted some of the challenges associated with testing Web applications running on the cloud. Due to the shared nature of cloud platforms, extreme care must be exercised to validate the test environment. In particular, systematic experimentation needs to be done to eliminate bottlenecks extraneous to the server being tested so that the true performance influence of the cloud platform can be inferred. We believe that our work can serve as an example for others interested in testing the performance of cloud-based Web applications. Following our proposed test procedure, we studied the performance of three different types of Amazon EC2 Web server instances under various degrees of load. We studied the response time of these three instances on an hourly basis for seven days of a week. We observed significant performance variations in the long run for each type of instance.

Future work will include running experiments with a more diverse set of workloads. Furthermore, we need to run experiments for a longer period of time, *e.g.*, a couple of months, to see whether the performance variability we have observed over a week is consistent or not. Future work will also focus on identifying the root cause for the observed performance variability in EC2. In the future, we will develop detection techniques to distinguish performance problems arising due to platform issues as opposed to expected performance variations caused by workload fluctuations. We will also focus on testing other cloud provider platforms such as Google App Engine, Heroku and Windows Azure.

VII. ACKNOWLEDGMENTS

This work was funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] G. Kousiouris, T. Cucinotta, and T. Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *J. Syst. Softw.*, 2011.

[2] J. Mukherjee, D. Krishnamurthy, J. Rolia, and C. Hyser, "Resource contention detection and management for consolidated workloads," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 2013.

[3] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011.

[4] R. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "An early performance analysis of cloud computing services for scientific computing," *TU Delft, Tech. Rep., Dec 2008, [Online] Available*.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945462

[6] "collectl- linux man page." [Online]. Available: http://linux.die.net/man/1/collectl

[7] httperf, "Http performance measurement tool,." [Online]. Available: http://www.hpl.hp.com/research/linux/httperf/httperf-man-0.9.pdf

[8] "Iperf- the tcp/udp bandwidth measurement tool." [Online]. Available: http://iperf.fr/

[9] J. Gao, X. Bai, and W.-T. Tsai, "Cloud testing-issues, challenges, needs and practice," *Software Engineering: An International Journal*, vol. 1, no. 1, pp. 9–23, 2011.

[10] L. Riungu, O. Taipale, and K. Smolander, "Research issues for software testing in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010.

[11] A. Turner, A. Fox, J. Payne, and H. Kim, "C-mart: Benchmarking the cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, 2013.

[12] J. Kiruthika and S. Khaddaj, "System performance in cloud services: Stability and resource allocation," in *Distributed Computing and Applications to Business, Engineering Science (DCABES), 2013 12th International Symposium on*, 2013.

[13] "Cloudharmony." [Online]. Available: http://cloudharmony.com