2011 IEEE International Parallel & Distributed Processing Symposium

Evaluating Load Generation in Virtualized Environments for Software Performance Testing

Marco A. S. Netto[1], Suzane Menon[1], Hugo V. Vieira[1], Leandro T. Costa[1],
Flavio M. de Oliveira[1], Rodrigo Saad[2], Avelino Zorzo[1]
[1]*Pontifical Catholic University of Rio Grande do Sul (PUCRS) – Faculty of Informatics*
[2]*Dell Inc.*
Porto Alegre, Brazil

***Abstract*—Before placing a software system into production, it is necessary to guarantee it provides users with a certain level of Quality-of-Service. Intensive performance testing is then necessary to achieve such a level and the tests require an isolated computing environment. Virtualization can therefore play an important role for saving energy costs by reducing the number of servers required to run performance tests and for allowing performance isolation when executing multiple tests in the same computing infrastructure. Load generation is an important component in performance testing as it simulates users interacting with the target application. This paper presents our experience in using a virtualized environment for load generation aimed at performance testing. We measured several performance metrics and varied system load, number of virtual machines per physical resource, and the CPU pinning schema for comparison of virtual and physical machines. The two main findings from our experiments are that physical machines produced steadier and faster response times under heavy load and that the pinning schema is an important aspect when setting up a virtualized environment for load generation.**

***Keywords*-Software Performance Testing; Virtualization; Load Generation; LoadRunner; Multi-core Architecture.**

I. Introduction

Software performance testing has become an important research area aimed at investigating how software systems behave under different loads and computing facilities. Not only companies may be affected by using software systems that provide users with low Quality-of-Service, but also mission critical systems may cause catastrophic results when not performing well.

Software systems are tested using various input parameters and analysed by several metrics. In addition, complex applications may have individual components that are tested separately. Therefore, computing power and performance isolation [1], [2] for each test become essential. The first requirement can be met with an off-the-shelf cluster, whereas the second one with virtualization. Besides performance isolation, virtualization can save energy costs by reducing the number of physical machines used for performance tests.

As several tests are required before deploying a software into production, most of the research on testing for virtualized environments have focused on automating test deployment [3]–[6]. Although the deployment phase is relevant, it is important to understand the impact of virtualization [7] when performing these tests. One of the main components that may suffer from virtualization is the load generator. This component is responsible for simulating users contacting the target application that needs to be tested.

Little work has been done to understand load generation on virtualized environments [4]. This paper presents our experience in using a virtualized environment for load generation aimed at performance testing. We measured several metrics, including throughput, response time, and transactions per second for comparison of virtual and physical machines. We also varied the system load, number of virtual machines per physical resource, and the pinning schema for distributing CPUs to the virtual machines (VMs). The two main findings from our experiments are that physical machines produced steadier and faster response times under heavy load and that the CPU pinning schema is an important aspect when setting up a virtualized environment.

The remainder of the paper is organized as follows. Section II introduces relevant concepts of software performance testing; Section III describes the main bottlenecks when using virtualization for load generation; Section IV presents the evaluation of load generation under various scenarios; Section V discusses related work in virtualization and software performance testing; and Section VI concludes the paper with the main findings and future research directions.

II. Software Performance Testing

With the growing demand to serve a greater number of customers, there has been an increasing number of web services and applications to allow multiple users to access simultaneously the system resources. In order to analyze the capacity of these systems and maintain their availability and performance as expected, it is necessary to simulate user activities under several conditions. In this context, through software performance testing, it is possible to develop effective strategies to maintain system performance at an acceptable level [8].

These tests aim at verifying whether the system performance is in line with design expectations, subjecting it to a certain load at a given computing environment [9], [10].

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.244

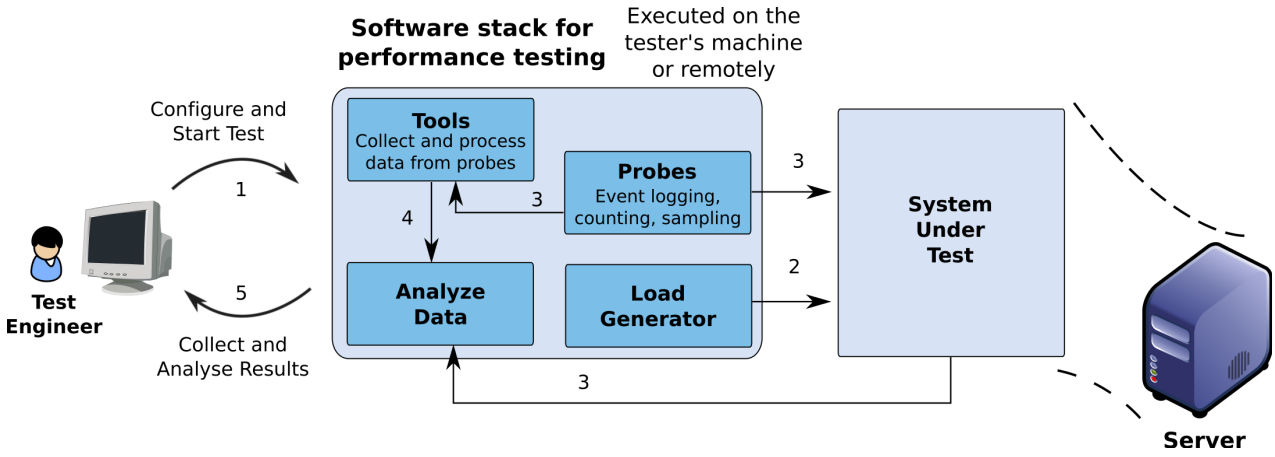


Figure 1. Software stack for performance testing and its interactions with the user and target application.

In order to execute these tests, several workload types and computing environments can be configured to evaluate the system under various conditions, being therefore able to identify possible bottlenecks that prevent the system to meet users' expected Quality-of-Service defined in Service Level Agreements. Moreover, besides identifying bottlenecks, performance testing helps to determine the required time for software to perform a task and to provide an idea about how stable the system is through a load change [11].

A. Performance Testing Architecture

Software engineers usually adopt automated testing to evaluate system performance due to the large number of users accessing a target application simultaneously. Thus, a machine set that represents simulated users who access the target application is defined. Actions performed by these users are called workloads, which trigger requests (e.g. HTML/HTTP) to the server where the application is hosted.

The software performance testing comprises several steps and modules. Figure 1 summarizes the software stack for performance testing [8] and its interaction with the test engineer and the target application. The test engineer starts configuring the test and triggering the load generator (Step 1). The latter applies the load to the system under test (Step 2). While the system is executing, the Probes monitor the resources using tools such as SiteScope[1] and perfmon[2] , meanwhile the data is then separated for further analysis (Step 3). Once the data is available and the execution completed, the tools collect and process the data from the probes (Step 4), which is then used by the engineer for analysing the system performance (Step 5).

After the test, the test engineer can then track how resource and system characteristics vary depending on workload. Among these characteristics, the most relevant are the following [12], [13]:

- Availability: time that an application is available for user;
- Response time: time that an application takes to respond to a request;
- Throughput: amount of processed data in a given period of time;
- Use: system resources used by the application (e.g. disk, network, memory, and CPU utilization).

B. LoadRunner

LoadRunner [14] is a tool used in performance testing to simulate several users accessing the target application. These users are configured to reproduce real workloads that are used in the production environment. In order to execute a performance test using LoadRunner, it is necessary to perform the following steps (see Figure 2 [14]):

Figure 2. Test steps using LoadRunner.

- Test plan: defines the requirements for performance testing, for example, expected response time for a given number of simultaneous users;
- VUser script creation: records the user activities on the application and generates scripts automatically;
- Scenario definition: configures the test environment using the LoadRunner Controller. It is possible to define input parameters, such as the number of users, testing time, ramp-up and ramp-down times;
- Scenario execution: manages and monitors the load test using the LoadRunner Controller;
- Result analysis: shows the test results using the LoadRunner Analyzer tool.

[1]SiteScope: http://sitescope.tellurian.net/SiteScope/docs/UserGuide.htm
[2]Performance monitoring tool for Windows: http://msdn.microsoft.com.

In order to perform these steps, LoadRunner has three tools: Virtual User Generator (VUGen), which is responsible for capturing user information, accessing the target application, and then generating a performance test script, also known as the *virtual user script*; Controller, which is responsible for managing, configuring, and monitoring load testing, which is executed from the script generated by the VUGen tool; and Analyser, which creates reports based on test results—these reports contain graphics and values corresponding to metrics monitored and managed by Controller.

III. Load Generation on Virtualized Environments

As mentioned before, the load generator assigns load to the target application. Here we describe in details how the load generator works and issues on how this module is placed in a VM.

The users' behaviour used by the load generator is defined through the *test script*. In order to generate this script, a tool is used to record all transactions executed by the simulated user on the target application. This script is used to reproduce the user actions and is composed of three parts: $vuser\_init$ executes when the virtual user is initialized; $Action$ contains sequences of activities that are executed along with the test; and $vuser\_end$ contains the actions that are executed when the virtual user finalizes its execution. Algorithm 1 presents an example of a script pseudo-code that simulates a user accessing a shopping website. The user initially contacts the server (Line 5), selects items (Line 7), and pays for them (Line 10). A typical load generator allows the inclusion of a simulated user think time (Lines 6 and 8), which can be a static number or a function that generates random numbers according to a given distribution.

Algorithm 1: Pseudo-code for the test script.

```
1  begin
       /* vuser_init                                   */
2      constructor procedures
3  end
4  begin
       /* Actions list                                 */
5      connect_to_application_server(server)
6      think_time(t_1)
7      select_item(item_1)
8      think_time(t_2)
9      ...
10     pay_items(items[])
11     ...
12 end
13 begin
       /* vuser_end                                    */
14     destructor procedures
15 end
```

After the script is generated, the test engineer configures the test scenario using several parameters such as number of threads that simulate users accessing the application, startup time, total test execution time, features and metrics that need to be monitored. Thereafter, the test can be performed. Once all the monitored data is collected, test results can be analyzed, using graphics to determine relationships and application behavior during the test.

Load generators can be placed into VMs in order to achieve performance isolation when multiple experiments have to be executed at the same time in the same infrastructure. By using virtualization, each experiment can have a portion of the computing resources, which can be accessed in an isolated fashion; such a mechanism can be hardly achieved using the physical resources directly. From the deployment's perspective, there is no difference between triggering load generators in virtual and physical machines; only the IP address has to be configured. However, from the performance's perspective, one of the main drawbacks is the overhead imposed by the virtualization layer.

The virtualization overhead comes mainly from *I/O* operations, because these operations of the VM operating systems are interpreted and translated by the hypervisor that passes them to **Dom-0**, which is responsible for accessing and processing all requests for VMs' hardware. Thus, the overall system performance is affected. Dom-0 is a privileged domain, which is a VM with direct access to the actual hardware.

One way to minimize the effect of overload due to the operations of *I/O* is to increase the CPU power of the Dom-0, because with a greater amount of resources the Dom-0 is capable of managing the requests of the VMs in a more optimized way. For example, in a quad-core machine, one can have a single core dedicated to Dom-0, instead of sharing with all cores of the VMs. This can be achieved by **pinning** the CPU, which specifies the virtual CPUs (VCPUs) are mapped to physical CPUs. The evaluation section contains experiments that show the impact of multiple CPUs pinning schemes.

IV. Evaluation

This section presents the environment setup used in our experiments. It also describes the metrics, experiment parameters, and result analysis. The aim of the experiments is to show the performance difference between using physical and virtual machines, and the impact of CPU pinning schema when setting up the virtualized environment.

A. Environment Setup

The infrastructure used to perform the experiments consists of four machines Dell PowerEdge R610: 2 Intel Xeon E5520 Quadcore Processors and 16GB RAM. One machine executes the application to be tested, two machines are used as load generators (one physical and the other with a set

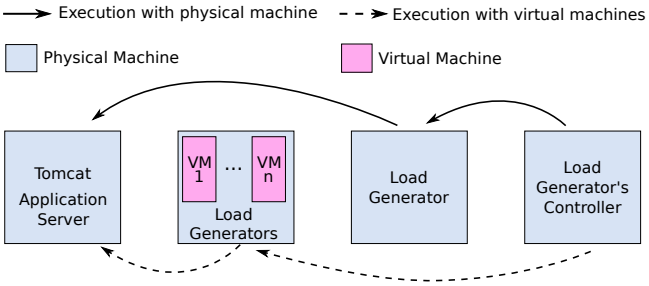


Figure 3. Experiment setup.

Table I
LIST OF SOFTWARE PACKAGES AND THEIR RESPECTIVE VERSIONS.

Software	Version
LoadRunner	9.5
Oracle VM Server	2.2
Xen	3.4
Windows Server	2008 (32-bits)
Tomcat	5.5.29

of virtual machines), and one that contains the controller for trigging and managing the tests. Figure 3 illustrates the information flow among the machines involved in the experiment.

All machines run Windows Server, except Dom-0 (*i.e.* Xen [15]), which runs Linux as it is based on Oracle VM Server. As we wanted to evaluate the use of virtualization for load generation, we chose a lightweight application (*i.e.* the Tomcat application server) to be able to easily increase resource consumption for the generators. The load generator script accesses Tomcat web pages and compacts a 5MB file. Therefore, we have an experimental setup that uses disk, CPU, and network. Table I summarizes the list of software packages and their respective versions.

We varied three input parameters:

- **Number of users:** these are the virtual/simulated users that influence the load assigned to the application: 25, 50, 100, and 150;
- **Number of VMs:** the number of virtual machines on the physical host: 1, 2, and 4;
- **CPU pinning schema:** the number of CPUs dedicated to Dom-0.

We measured almost 30 metrics, which include throughput, response time, pages downloaded per second, and several system metrics, such as memory, CPU, and network consumption. We show the results for some of these metrics, whereas the other ones are used to enhance our analysis. In addition, each experiment was executed for two hours. The graphs presented in the following section include the average result of a given metric with its respective standard deviation represented by vertical bars.

B. Results and Analysis

1) Physical versus virtual machine: The first set of experiments compares performance results from one physical machine against one VM. For these experiments, all CPUs are shared between the VM and the Dom-0. Figure 4 presents response time and throughput for both physical and virtual machine. For response time, we observe that both average and standard deviation increase with the number of users in the system, whereas for the physical machine these values are steady for different system loads. This happens because of the *I/O* overhead imposed by the virtualization layer.

In order to verify the actual bottleneck from *I/O* operations in virtualization, we measured system resource metrics and present them in Tables II, III, and IV, which represent CPU usage, disk queue length, and network usage, respectively. CPU usage increases for both virtual and physical machines with the increase of load applied to the system. However, it is interesting to remark that the physical machine has a higher CPU usage. This happens because data is available sooner for the physical machine than for the virtual one.

The main bottleneck observed is the disk access. Table III shows that even though the load increases with the number of users, the disk queue length remains steady after a certain threshold, *i.e.* 40 and 25 for virtual and physical machine, respectively. This difference between these two values has a considerable impact on the overall system performance, as observed in the response time metric (Figure 4) and the network usage (Table IV), as more data is ready to be transferred using the physical machine.

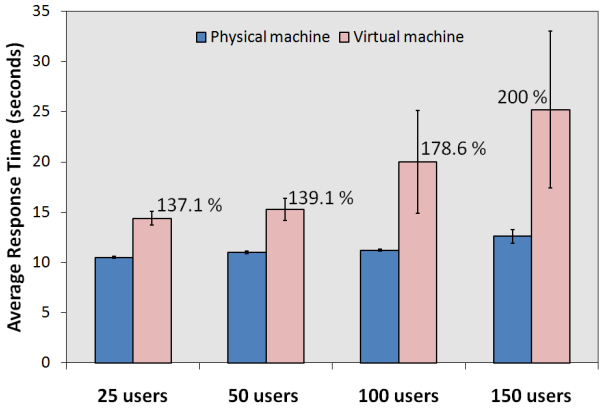


Figure 4. Physical versus virtual machine.

2) Impact of CPU pinning: As observed in the previous section, *I/O* operations generate overhead in the VMs. Here we present results for three CPU pinning schemas: (i) all CPUs are shared between the virtual and Dom-0 (**non-dedicated CPUs**); (ii) 2 CPUs are dedicated to Dom-0 (**2 dedicated CPUs**); and (iii) 4 CPUs are dedicated to Dom-0 (**4 dedicated CPUs**). For the last two cases, the remaining

Table II
PERCENTAGE OF THE AVERAGE CPU USAGE.

Users	25	50	100	150
Values for VM	7.3	14.8	25.3	34.3
Values for phys. mach.	6.7	13.4	29.3	48.1

Table III
AVERAGE DISK QUEUE LENGTH (# OF REQUESTS).

Users	25	50	100	150
Values for VM	40.4	41.4	39.6	38.6
Values for phys. mach.	6.3	24.9	24.1	24.5

Table IV
AVERAGE BYTES RECEIVED PER SECOND (10^3).

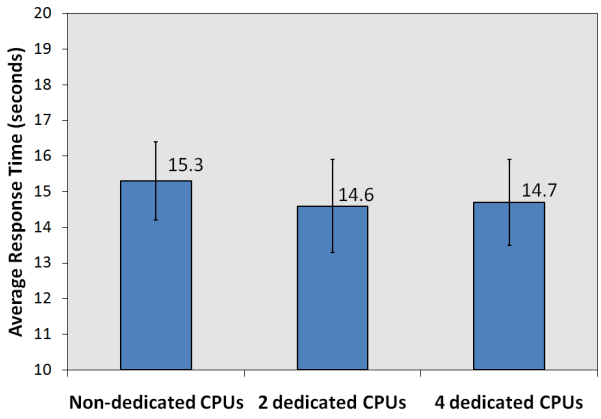
Users	25	50	100	150
Values for VM	78.3	147.4	238.2	282.7
Values for phys. mach.	107.2	203.6	398.6	529.4

CPUs are shared between the VM and Dom-0. Here we present the results for 50 and 150 users.

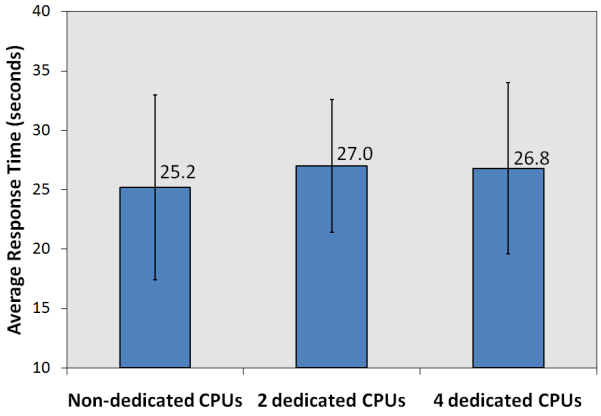
Figures 5 and 6 depict the results of response time and throughput respectively. We observe that different CPU pinning schema have different impact depending on system load and metric. For instance, dedicating 2 CPUs to Dom-0 for 50 users has a reduction of approximately 5% compared to non-dedicated CPU schema, whereas for 150 users, sharing all CPUs with the VMs has a reduction of approximately 8% compared to dedicating CPUs to Dom-0. For throughput, the most significant difference is for 150 users, in which no dedicated CPUs to Dom-0 increases in 10% the throughput compared to 2 dedicated CPUs. For the case of 150 users, more CPU is required compared to 50 users (Table II), therefore providing as much CPU as possible to the VM produces a higher throughput. However, for 50 users, increasing CPU power in Dom-0 makes writing disk speed increase in 10%, which results in higher throughput.

Increasing the CPU power of Dom-0 enhances the performance of *I/O* operations, but decreases the CPU performance inside the VM. We confirmed the improvement by measuring two disk-related metrics: average disk data writing time and average disk queue length, showed in Tables V and VI, respectively. The main reduction comes from changing from non-dedicated to 2 dedicated cores to Dom-0. As the 50-user scenario is not as CPU demanding as as the 150-user one, the latter requires more CPU for the VMs. Therefore, not including the dedicated cores to Dom-0 produces faster response time for 150 users (Figure 5).

3) Multiple VMs per physical machine: This section shows results of increasing the number of VMs in the physical machine considering the average system load scenario, *i.e.* 50 users. In order to keep the same load for each



(a) 50 users.



(b) 150 users.

Figure 5. Impact of CPU pinning on response time.

Table V
AVERAGE DISK DATA WRITING TIME (SECONDS).

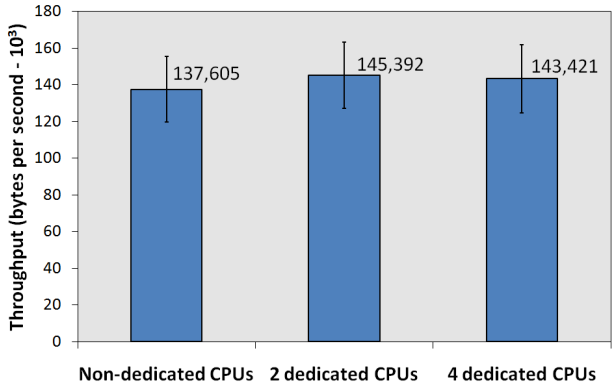
CPU Pinning	non-ded. cores	2 ded. cores	4 ded. cores
50 users	0.80	0.71	0.71
150 users	0.93	0.71	0.70

Table VI
AVERAGE DISK QUEUE LENGTH (# OF REQUESTS).

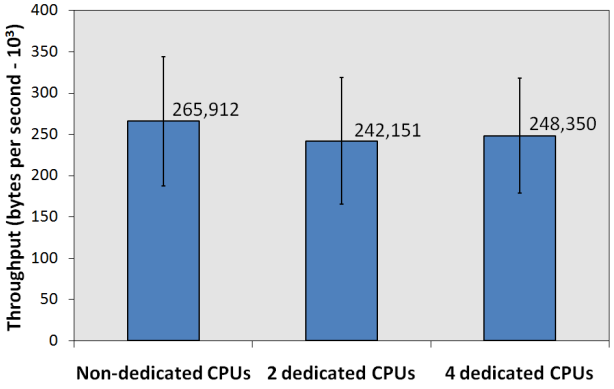
CPU Pinning	non-ded. cores	2 ded. cores	4 ded. cores
50 users	41.4	41.4	40.5
150 users	47.6	39.3	39.1

experiment, we split the number of users among the VMs. The metrics evaluated in these experiments are response time and throughput.

Apart from the physical machine producing faster response times and higher throughput compared to VMs, these perform worst and are more unstable when the number of VMs increases. This behaviour can be easily seen in Figures 7 and 9, which show results for response time and trans-



(a) 50 users.



(b) 150 users.

Figure 6. Impact of CPU pinning on throughput.

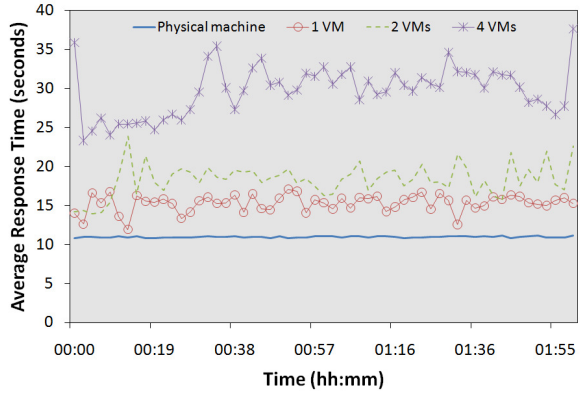


Figure 7. Impact of multiple VMs on response time - 50 users.

action response time percentile, respectively. This happens because Dom-0 requires more CPU to handle multiple VMs. For instance, for one VM, the CPU usage to handle 50 users is 14.75%, whereas for four VMs the usage is 7.2%. This reduction is due to Dom-0 competing for CPU with the VMs.

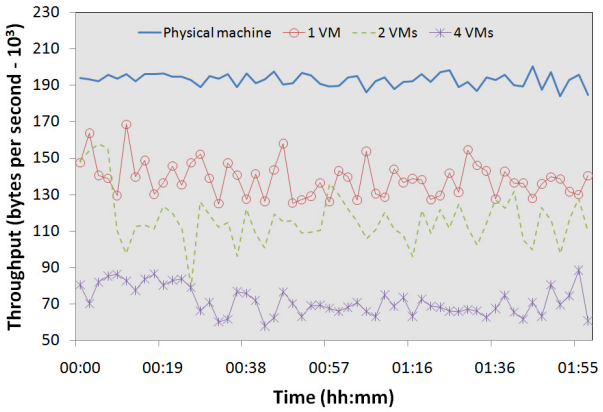


Figure 8. Impact of multiple VMs on throughput - 50 users.

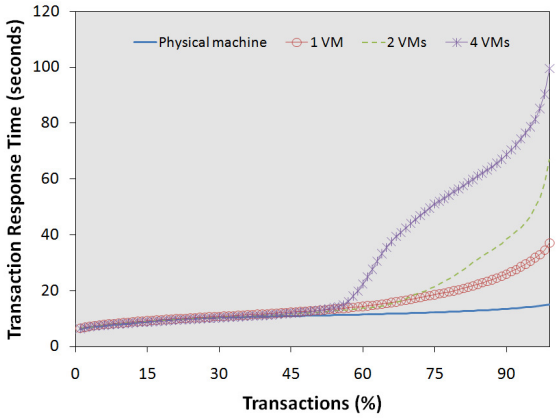


Figure 9. Impact of multiple VMs on transaction response time percentile - 50 users.

Table VII
AVERAGE PROCESSOR TIME AND DISK QUEUE LENGTH FOR MULTIPLE VMS.

Number of VMs	1 VM	2 VMs	4 VMs
Avg. processor usage (%)	14.8	11.7	7.2
Disk queue length	41.4	38.5	39.8

4) Reduzing I/O operations: The results presented so far are based on simulated users who perform file compression, which is an *I/O* intensive operation. By using such a script we observed the limitations of VMs in comparison to physical machines for load generation. We also performed a set of experiments with another script, which does not contain the compression phase, but only web page requests to the application server. Figure 10 summarizes the response time for the virtual and physical machine varying the number of simulated users who access the server. For comparison reasons, the results with file compression in this figure represent only the time to execute the web requests, *i.e.* the

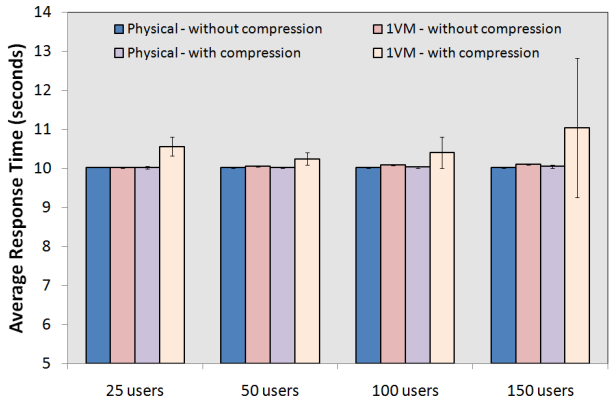


Figure 10. Response time comparison with reduced number of *I/O* operations.

file compressing time is not included. We observe that when removing the *I/O* intensive phase of the script, the results of the virtual and physical machines are similar, which shows that for a non *I/O* intensive load generation script, VMs are a good alternative for replacing physical machines.

V. Related work

This paper fits into the following main research topics: (i) performance comparison of virtual and physical machines; and (ii) use of virtualized environments for software performance testing. This section presents some of these studies and their main differences compared to our work.

Our experiments show the importance of CPU pinning for performance metrics. Somani and Chaudhary [16] go further and study the need of dynamically balancing the load in a physical server. Moreover, they propose an implementation of Global Load Balancing algorithm and use the CPU pinning mechanism provided by Xen. Cherkasova and Gardner [17] investigate CPU overhead produced by *I/O* operations using Xen VM Monitor. When applied 100% CPU, they observed a 26.5% CPU usage for Dom-0, which is a similar result we got in our experiments. There are also other works that try to minimize the *I/O* overhead imposed by virtualization, for instance, Appavoo *et al.* [18] presented a mechanism to improve performance of network access in virtualized machines, and Wei *et al.* [19] investigated the use of dedicated Xen domains to handle *I/O* operations. Liu and Abali [20] proposed the Virtualization Polling Engine (VPE) that uses dedicated CPU cores to help with the virtualization of *I/O* devices by using an event-driven execution model with dedicated polling threads. Our paper shows that a proper CPU pinning has an impact on performance metrics, which relates to the limitation of the virtualization technology to handle *I/O* operations.

Several projects have started to explore virtualization for software testing. Banzai *et al.* [3] developed D-Cloud for software performance testing using cloud computing technology and VMs in order to reduce cost and time. Their focus is mainly on fault tolerance testing. Gaisbauer *et al.* [4] designed and implemented the Virtualization-aware Automated Testing Service (VATS). The aim of the VATS framework is to automate test execution in Cloud computing environments. Duro *et al.* [5] introduced the VIRTU tool aimed at managing, configuring, and testing applications in virtualized environments. Kim *et al.* [6] also proposed a middleware for performance testing for those environments.

Different from the existing works in software performance testing presented in this section, our aim is to focus not on the deploying phase of tests, but on the cost-benefits of using VMs for software performance testing. We described detailed experiments varying several parameters and analysing various metrics.

VI. Concluding Remarks and Future Work

Performance testing for software systems is essential to increase companies' profit and reduce risks in mission critical systems. Several tests are required before placing a system into production, and hence the testing process becomes time consuming and costly. Virtualization has become an important tool therefore to assist in the software testing because it enables savings in energy costs through server consolidation and performance isolation for multi-test scenarios. This paper has then presented our experience using load generation in a virtualized environment. Load generation is an important component of software testing since it emulates users' behaviour when accessing the target application. Although virtualization brings benefits, it comes with an overhead cost, mainly for *I/O* operations. We have thus analysed the overhead in several scenarios: changing system load, CPU pinning schema, and number of VMs per physical machine.

The main source of bottlenecks comes from the file compressing part (*i.e.* the *I/O* intensive part) of our load generator script, which requires considerable disk usage consumption. This makes the disk queue length of the VM up to six times longer than in the physical one in some scenarios. Another source of overhead comes from Dom-0 managing multiple VMs, as there is the context switching that consumes CPU. The CPU pinning then becomes an important aspect to be configured, as it determines how much CPU should be assigned to the VM manager. As observed in our experiments through the disk queue length metric, assigning more CPU to the manager reduces the *I/O* overhead, providing faster response times and higher throughput. The drawback is that, CPU is then removed from the VMs to perform CPU-intensive tasks. Therefore, there is a complex trade-off that has to be evaluated for each scenario. From our experiments, thus, the main findings of this paper are that physical machines produce steadier and better results under heavy load and that the CPU pinning schema

is an important configuration when setting up a virtualized environment for load generation. For the load generator script without the *I/O* intensive phase, we observed that VM is a good alternative for replacing physical machines. Therefore, test engineers should be careful when analyzing performance data from virtual machines and tuning the computing environment, especially for scenarios with heavy *I/O* operations.

One of the main future directions from this work is to define a system overhead model that can be used for software performance tests. Several companies have been adopting virtualization technologies to reduce costs, mainly associated with the number of servers used to perform tests. Therefore, as many overhead issues cannot be eliminated to compete with executions based on physical machines, it is important to have a system overhead model that can be used along with the software testing analysis, which would then allow more reliable results on virtualized environments.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for their comments on the paper. The experiments were performed in the High Performance Computing Lab at Catholic University of Rio Grande do Sul (LAD-PUCRS), Brazil, in partnership with Dell Brazil. Avelino F. Zorzo is a researcher supported by CNPq/Brazil. This work has also support from FAPERGS/CNPq.

REFERENCES

[1] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the Workshop on Experimental computer science (ExpCS'07)*. ACM, 2007.

[2] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, "Performance implications of virtualizing multicore cluster machines," in *Proceedings of the 2nd Workshop on System-level Virtualization for High Performance Computing (HPCVirt'08)*, ser. HPCVirt '08. ACM, 2008.

[3] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID'10)*. IEEE Computer Society, 2010.

[4] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia, "VATS: Virtualized-Aware Automated Test Service," in *Proceedings of the 5th International Conference on Quantitative Evaluation of Systems (QEST'08)*, 2008.

[5] N. Duro, R. Santos, J. ao Lourenço, H. Paulino, and J. ao Martins, "Open virtualization framework for testing ground systems," in *Proceedings of the 8th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD'10)*. ACM, 2010.

[6] G. hun Kim, H. choun Moon, G.-P. Song, and S.-K. Shin, "Software performance testing scheme using virtualization technology," in *Proceedings of the 4th International Conference on Ubiquitous Information Technologies Applications (ICUT'09)*. IEEE, 2009.

[7] U. Drepper, "The cost of virtualization," *Queue*, vol. 6, pp. 28–35, January 2008.

[8] I. Burnstein, *Practical software testing: a process-oriented approach*. Springer-Verlag, 2003.

[9] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," *IEEE Transactions on Software Engineering (TSE'00)*, vol. 26, no. 12, pp. 1147–1156, dec. 2000.

[10] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *Proceedings of the Future of Software Engineering (FOSE'07)*. Washington, DC, USA: IEEE Computer Society, 2007.

[11] G. J. Myers and C. Sandler, *The Art of Software Testing*, 2nd ed. John Wiley & Sons, June 2004.

[12] J. Z. Gao, J. Tsao, Y. Wu, and T. H.-S. Jacob, *Testing and Quality Assurance for Component-Based Software*. Norwood, USA: Artech House, Inc., 2003.

[13] I. Molyneaux, *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*, 1st ed. O'Reilly Media, Inc., 2009.

[14] R. Zheng, H. Wang, and Y. Pang, "Research on bio-inspired multi-net paralleling mechanism based on web application," in *Proceedings of the 7th international conference on Computational Science (ICCS'07)*. Springer-Verlag, 2007.

[15] 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," in *Proceedings of the 9th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, 2003.

[16] G. Somani and S. Chaudhary, "Load Balancing in Xen Virtual Machine Monitor," *Contemporary Computing*, pp. 62–70, 2010.

[17] L. Cherkasova and R. Gardner, "Measuring CPU overhead for I/O processing in the Xen virtual machine monitor," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. USENIX Association, 2005.

[18] J. Appavoo, A. Waterland, D. Da Silva, V. Uhlig, B. Rosenburg, E. Van Hensbergen, J. Stoess, R. Wisniewski, and U. Steinberg, "Providing a cloud network infrastructure on a supercomputer," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. ACM, 2010.

[19] J. Wei, J. R. Jackson, and J. A. Wiegert, "Towards scalable and high performance i/o virtualization - a case study," in *Proceedings of the High Performance Computation Conference (HPCC'07)*, 2007.

[20] J. Liu and B. Abali, "Virtualization polling engine (vpe): using dedicated cpu cores to accelerate i/o virtualization," in *Proceedings of the 23rd International Conference on Supercomputing (ICS'09)*. ACM, 2009.