Quantifying the Performance Isolation Properties of Virtualization Systems

Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane,
Demetrios Dimatos, Gary Hamilton, Michael McCabe

Clarkson University
{jnm, huwj, hapuarmg, deshantm, dimatosd, hamiltgr, mccabemt}@clarkson.edu

Abstract

In recent years, there have been a number of papers comparing the performance of different virtualization environments for x86. These comparisons have typically quantified the overhead of virtualization for one VM compared to a base OS. It has also been common to present data on the scalability of the system in terms of how many identically configured virtual machines can be run on a single physical machine or the performance degradation experienced when multiple VMs are running the same workload. . However, one important aspect of comparing virtualization systems is often not quantified – the degree to which they limit the impact of a misbehaving virtual machine on other virtual machines. Quantifying performance isolation is challenging because the impact can vary based on the type of "misbehavior". Nevertheless, performance isolation is an essential feature especially in commercial service hosting environments, a key target application of modern virtualization systems, where mutually untrusting VMs run on the same physical machine. In this paper, we present the design of a performance isolation benchmark that quantifies the impact on well-behaved VMs of various categories of extreme resource consumption in a misbehaving VM. Our test suite includes six different stress tests - a CPU intensive test, a memory intensive test, a disk intensive test, two network intensive tests (send and receive) and a fork bomb. We describe the design of our benchmark suite and present results of testing three flavors of virtualization – VMware (an example of full virtualization), Xen (an example of paravirtualization) and Solaris containers (an example of operating system level virtualization). We find that the full virtualization system offers complete isolation in all cases and that the paravirtualization system offers nearly the same benefits – no degradation in many cases with at most 1.4% degradation in the disk intensive test. The operating system level virtualization system experiences severe degradation for the memory intensive test and for the fork bomb test, no degradation on the CPU intensive test and mild degradation of 1.2 – 4% for the other tests. Our results highlight the difference between these classes of virtualization systems as well as the importance of considering multiple categories of resource consumption when evaluating the performance isolation properties of a virtualization system.

1. Introduction

Virtualization environments can be used for many different purposes. For example, virtualization can be used to maintain multiple software environments on the same host for testing or simply to allow a desktop user to run multiple operating systems on the same physical host. Virtualization environments have long been used in commercial server environments on platforms such as IBM's VM/370 [1] or zOS. Increasingly, virtualization environments for x86 platforms are targeting commercial server environments as well. [2, 3] In such an environment, a provider may allow multiple customers to administer virtual machines on the same physical host.

In recent years, there have been a number of papers comparing the performance of different virtualization environments for x86 such as Xen, VMware and UML [4] [5] [6]. These comparisons have focused on quantifying the overhead of virtualization for one VM compared to a base OS. In addition, researchers have examined the performance degradation experienced when multiple VMs are running the same workload. This is an especially relevant metric when determining a systems' suitability for supporting commercial hosting environments– a key target environment for many virtualization systems. In such an environment, a provider may allow multiple customers to administer virtual machines on the same physical host. It is natural for these mutually untrusting customers to want a certain guaranteed level of performance regardless of the actions taken by other VMs on the same physical host.

Thus, in a commercial hosting environment, there is another important aspect to the comparison that has received less attention – how well do different virtualization environments protect or isolate one virtual machine from another? Running in parallel with other web server VMs is quite different than running in parallel with a fork bomb or other resource hogs. Knowing the performance degradation in each case is important for both providers and customers of commercial hosting services.

In this paper, we present the design of a performance isolation benchmark and use it to examine three virtualization environments – VMware (an example of full virtualization), Xen (an example of paravirtualization) and Solaris containers (an example of operating system level virtualization). Our results highlight the difference between these classes of virtualization systems as well as the importance of considering multiple categories of resource consumption when evaluating the performance isolation properties of a virtualization system.

2. A Performance Isolation Benchmark Suite

To quantify the performance isolation of a virtualization system, we designed a test suite including six different stress tests - a CPU intensive test, a memory intensive test, a disk intensive test, two network intensive tests (send and receive) and a fork bomb.

To perform each test, we start a set of behaving virtual machines on the same physical machine. In our testing, we used web server virtual machines as an example of an important class of service VMs that might typically be deployed in a production environment.

We establish a baseline response time for this baseline configuration and then we introduce a stress test into one of the virtual machines. We quantify the performance degradation on both the misbehaving and well-behaved VMs.

In our testing, we used response time as reported by the SPEC web benchmark as the performance metric of interest. However, our stress test suite could be used in any production environment to assess the impact on other metrics of interest for the services running in the systems' virtual machines.

We are providing the source code for each of our stress tests along with instructions for compiling and running each one. The archive file containing the test suite also contains a variety of scripts we found useful in running the tests.

Table 1 summarizes the actions taken by each test.

Test	Description
Memory Intensive	Loops constantly allocating and touching memory more memory without freeing it.
Fork Bomb	Loops creating new child processes.
CPU Intensive	Tight loop containing both integer and floating point operations.
Disk Intensive	Running 10 threads of IOzone each running an alternating read and write pattern (iozone -i0 -i1 -r 4 -s 100M -t 10 -Rb).
Network Intensive (Transmit)	4 threads each constantly sending 60K sized packets over UDP to external receivers.
Network Intensive (Receive)	4 threads which each constantly read 60K packets over UDP from external receivers

Table 2: Description of Individual Stress Tests

3. Experience With Our Performance Isolation Benchmark Suite

In this section, we describe our experience with using our benchmark suite to test the performance isolation characteristics of Xen, VMware and Solaris containers.

3.1. Baseline Data

Before beginning our stress testing, we established some baseline data. Specifically, we ran SPECweb 2005 with 4 web servers each in their virtual machine. The server VMs were all hosted on a single IBM

ThinkCentre with Pentium 4 processor, 1 GB of memory and a gigabit Ethernet card. We used three different virtualization environments - Xen 3.0 stable, VMware Workstation 5.5 and Open Solaris 10. With Xen and VMware, we used the same Linux server image with Linux (2.6.12 kernel) running Apache 2. For Solaris, each Solaris container was running Apache 2. In both Xen and VMware, we assigned each virtual machine 128 MB of memory.

In SPECweb, additional machines are used as clients. The machines serving as clients in our testing were also IBM Thinkcentres. We used a different physical client to connect to each server virtual machine. Unless otherwise noted, each of our physical clients presented a load of 5 simulated clients.

At this load, all four web server instances provided 100% good response time as reported by the SPECweb clients over 3 iterations. These baseline numbers illustrate that the machine is well configured to handle the SPECweb requests. In other words, we are not taxing the system with this load and any degradation in performance seen in the stress tests can be contributed to the stress test itself.

We emphasize that our benchmark suite could be used to quantify the impact on any set of virtual machines not just web servers and not just using SPECweb as the performance metric of interest.

We recommend that a system be configured as would be appropriate for a production environment using whatever number and types of virtual machines give good common case performance. The stress tests can then be run in one of the virtual machine to quantify the degree of performance isolation provided by the system. We report results as a percentage degradation from the baseline configuration. For web servers, using SPEC to report the percentage of requests that receive a response in an acceptable amount of time is an appropriate performance metric. However, other metrics such as throughput or total run time may be appropriate for other types of services.

3.2 Stress Tests

After completing the baseline measurements, we ran a series of tests that stress a variety of system sources including memory, process creation, CPU, disk I/O and network I/O. In these tests, we started web servers in all four virtual machines, as in the baseline tests, and then in addition ran the stress test in one of the server virtual machines.

3.2.1. Memory Consumption

The memory stress test loops constantly allocating and touching additional memory. When this stress test was run in one of the Solaris containers, none of the Solaris containers survived to report results. The test effectively shut down all virtual machines – misbehaving and well-behaved. In the Xen case, the misbehaving VM did not report results, but all others continued to report nearly 100% good results as before. In the VMware case, the misbehaving VM survived to report significantly degraded performance (8.6% good responses overall) and the other three servers continued to report 100% good response time, as in the baseline.

3.2.2. Fork Bomb

We used a classic fork bomb test that loops creating new child processes. Under both Xen and VMware, the misbehaving virtual machine presented no results, but the other three well-behaved containers continued to report 100% (or near 100%) good response time.

For Solaris, we tested in two configurations – with the default container setup and then again with an option we found that was described as avoiding problems with fork bombs in containers. [1]In the first case, results were not reported for any of the four containers. In the second case, results were reported, but severe degradation occurred. The good response rate averaged 10% across all four containers and interestingly the misbehaving container actually demonstrated the best results with 12% good response.

3.2.3. CPU Intensive Test

Our third test stresses CPU usage with a tight loop containing both integer and floating point operations. All three of our virtualization systems performed well on this test – even the misbehaving VMs. We verified on all platforms that the CPU load on the misbehaving server does rise to nearly 100%.

[1] As part of our experiments with Solaris containers, we explored various configuration options [13]. In the official Solaris documentation, we saw reference to both deny and none options for resource control values. [14] With the deny option in place, we were unable to boot the container and we subsequently found on various Solaris user groups that the deny option is not currently supported [11]. We did try the none option and somewhat to our surprise found that it showed a beneficial effect. Without the none option, a fork bomb would render the entire system unresponsive to even the keyboard, but with the none option we were able to Control-C the fork bomb and regain control. We posted asking if there was any better way to configure the container, but received no answer.

We suspect that the normal OS CPU scheduling algorithms are already sufficient to allow the web server sufficient CPU time. We wondered what would happen if we changed either the underlying OS and/or the guest OS. Towards this end we tried an additional experiment. We ran the same test on VMware running on Windows with the same Linux server images. In this case, performance was still excellent – 100% good performance on the normal VMs and 99% good in the misbehaving VM. It is interesting to note the slight, yet repeatable degradation is due to the change of CPU scheduling policy in the underlying OS from Linux to Windows.

3.2.4. Disk Intensive Test

For a disk intensive stress test, we chose not to write our own, but rather to use IOzone [9]. Specifically, we ran 10 threads of IOzone each running an alternating read and write pattern (iozone -i0 -i1 -r 4 -s 100M -t 10 -Rb). The results of this test were quite interesting.

For Solaris, as we've seen in other cases, the impact was similar on all four containers, but the performance degradation was minor but repeatable (1-2%).

On VMware, 100% good performance as maintained on the three well-behaved VMs. However, the misbehaving VM saw a degradation of 40%.

On, Xen, the situation was mixed. The misbehaving VM saw a degradation of 12% and the other three VMs were also impacted, showing an average degradation of 1-2%. With Xen's proposed hardware access model, a specialized device driver VM could be written to ensure quality of service guarantees for each client VM [10]

3.2.5. Network I/O Intensive Test

Our last set of stress tests involved a high level of network I/O. We examined both server transmitting and server receiving. For both sets of tests, we used other machines (not the SPECweb servers or clients) as the source or sink of the data.

3.2.5.1. Server Transmits Data

For the transmitting stress test, we started 4 threads which each constantly sent 60K sized packets over UDP to external receivers. For this test, the results were once again mixed.

All four Solaris containers showed degradations of about 4%. Under VMware, the well-behaved VMs continue to show 100% good response, but the misbehaving VM shows substantial degradation of 53%. For Xen, the well-behaved VMs show almost no degradation and the misbehaving VMs shows a slight but repeatable degradation of less than 1%.

3.2.5.2. Server Receives Data

Finally, for the receiving stress test, we started 4 threads which each constantly read 60K packets over UDP from external senders.

The results for this test were similar to the server transmit test on both Xen and Solaris. On Solaris, all four containers were impacted and the average degradation was about 2%. For Xen, there was a slight but repeatable degradation the misbehaving VM, but all the well-behaving VMs were unaffected.

For VMware, all four VMs retained 100% good response. We did not see the substantial degradation on the misbehaving VM that we saw in the sender transmit case. We suspect that in the face of network contention that the incoming packets are simply dropped before they impact any of the four web servers.

	VMware		Xen		Solaris	
	Good	***Bad***	***Good***	***Bad***	***Good***	***Bad***
Memory	0	91.30	0.03	DNR	DNR	DNR
Fork	0	DNR	0.04	DNR	90.03	87.80
CPU	0	0	0	0.03	0	0
Disk Intensive	0	39.80	1.37	11.53	1.48	1.23
Network Server Transmits	0	52.9	0.01	0.33	4.00	3.53
Network Server Receives	0	0	0.03	0.10	1.24	1.67

Table 2: Summary of Stress Test Results Percent of degradation in good response rate. For each test, the percent degradation for both the bad or misbehaving VM is shown, as well as, the average degradation across the three good or well-behaving VMs. DNR indicates the SPECweb client reported only an error and no actual results because of the unresponsiveness of the server it was testing.

3.6 Summary of Results

We collect our results in Table 2. For each test, we report the percent degradation in good response rate for both the misbehaving or bad VM and the average for the three well-behaving or good VMs.

One thing that is clear from Table 2 is the importance of considering multiple types of "misbehavior" when evaluating the performance isolation properties of a virtualization system. If we looked only at the CPU intensive test results, our conclusions would be different than if we considered the disk and network intensive tests or the memory intensive and fork bomb tests.

From the first column, it is clear that VMware completely protects the well-behaved VMs under all stress tests. Its performance is sometimes substantially lower for the misbehaving VM, but in a commercial hosting environment this would be exactly the right tradeoff to make.

Xen also protects the well-behaved VMs relatively well. The average degradation for the disk intensive case is the worst at 1.37%. One thing that this table highlights, however, is a slight but consistent degradation on most tests.

Solaris does not isolate well-behaved containers from misbehaving containers. For all tests, the well-behaved containers share the fate of the misbehaving one. This would be bad news indeed in a commercial hosting environment in which different customers are sharing the resources of a single physical host.

Solaris containers do have advantages that appear in other situations, like ease of creating a new VM and the ability to create more containers than would be possible with Xen or VMware (up to 8192) on the same system. However, this may in part lead to the resource isolation problem. If resources are committed when a VM is created, it is easier to guarantee those resources despite the actions of others. However, committing resources at creation time also limits the number of VMs that can be created. In an environment where all containers are under the same administrative control, this may be a reasonable trade-off.

On the other hand, operating system level virtualization systems by definition do not support the ability virtual machines with many different operating systems on the same physical machine. In many environments, this support for software heterogeneity is a key motivation to use a virtualization system. For example, maintaining a Windows XP VM and a Windows Vista VM and a RedHat Linux VM and a SUSE Linux VM to reduce the hardware requirements for testing software that runs on many platforms.

4. Conclusions

As virtualization systems for commodity platforms become more and more common, the importance of benchmarks that compare virtualization environments increases. The issue of isolation from misbehaving VMs is an important one to consider, especially for a commercial hosting environment.

We have demonstrated the importance of having a performance benchmark suite that considers multiple types of “misbehavior”. We have designed such as suite and present results we obtained from using it to compare VMware, Xen and Solaris containers.

Our results clearly highlight differences between major classes of virtualization systems – full virtualization like VMware, paravirtualization like Xen and operating system level virtualization like Solaris containers. Full virtualization completely protected the well-behaved VMs in all of our stress tests. Paravirtualization offers excellent resource isolation as well. In our Xen tests, the well-behaved VMs suffered at most a 4% degradation for the disk intensive test with many other tests showing only slight but repeatable degradation. With Solaris containers, the well-behaved containers shared the fate of the misbehaving containers in all cases.

Each of these environments could implement a high degree of resource isolation. There is nothing preventing strong resource isolation in any of these environments. Strong resource isolation could be implemented in any one of the systems we have studied. Even for operating level virtualization systems, the operating system could be modified to implement new resource scheduling algorithms that enforce resource isolation across containers for all resources: disk, network (incoming and outgoing), memory, CPU, etc. Solaris, for example, is working on adding more resource control into its container environment [11] and has provided APIs for the eventual controls, but it appears that users have been looking for these additions for at least a year. In addition, it appears that there has been work on similar technologies such as Sun’s Dynamic System Domains since 1996[12]. However, these results suggest that it is harder to retrofit isolation into a general purpose OS than to add it in clearly defined virtualization layers.

5. References

[1] R. Creasy IBM Journal of Research and Development. Vol. 25, Number 5. Page 483. Published 1981. The Origin of the VM/370 Time-Sharing System.

[2] K. Fraser, S. Hand, T. Harris, I. Leslie, and I. Pratt. The Xenoserver Computing Infrastructure. Technical Report UCAM-CL-TR-552, University ofCambridge, Computer Laboratory, Jan. 2003.

[3] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform, April 2003.

[4] A. Whitaker, M. Shaw, S. Gribble. Scale and Performance in the Denali Isolation Kernel. Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), , pages 195-210, Boston, MA, USA, December 2002.

[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. Proceedings of the 19th ACM symposium on Operating Systems Principles, pp 164-177, Bolton Landing, NY, USA, 2003

[6] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne and J. Matthews. Xen and the Art of Repeated Research. Proceedings of the USENIX 2004 Annual Technical Conference, FREENIX Track, pp. 135-144, June 2004.

[8] SPECweb 2005, http://www.spec.org/web2005, Accessed January 2007.

[9] IOzone, http://www.iozone.org, Accessed January 2007.

[10] K.Fraser, S.Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe Hardware Access with the Xen Virtual Machine Monitor, 1st Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure (OASIS-1), October 2004.

[11] Solaris Forums. Zoneadm- Why not action=deny in rctl?.URL http://forum.sun.com/thread.jspa?threadID=21712&tstart=0, Accessed January 2007.

[12] Sun Microsystems. Solaris Containers – Server Virtualization and Manageability, p. 5, September 2004.

[13] P.Galvin. Solaris 10 Containers, USENIX login, pp.11-14, October 2005.

[14] Sun Microsystems. Configuring Resource Controls and Attributes, URL http://docs.sun.com/app/docs/doc/817-1592/6mhahuoiq?l=en&a=view, Accessed January 2007.